APPENDIX B

```
/*****************************************************************************/
/*   Program Name      : Drive.c                                               */
/*   Description       : 4x 5.25" DSP Servo controller main kernal             */
/*   Part Number       : 562096                                                */
/*   Date              : 8/12/93                                               */
/*   O/S               : N/A                                                   */
/*   Compiler          : TI TMS320C2x/C5x Compiler.#TMDS3242855-02.Rel. 6.0 */
/*   Support Packages  : N/A                                                   */
/*   Author            : Dave Schell                                           */
/*   Required Files    : Drive.c.Interupt.asm.C50_init.asm.Seek.c.Drive.h      */
/*                     : Recal.c                                               */
/*   Hardware Required : Part # XXXXXX                                         */
/*   Install. Instr.   : Link in with Drive code                               */
/*   Operating Instr.  : N/A                                                   */
/*                                                                             */
/*   Rev History                                                               */
/*     Date     Rev  C#  Init   Change Description                             */
/*   4/14/94    XA   00   DLS   Initial Release                                 */
/*****************************************************************************/

#include "drive.h"

main ()
{
/* Debug start */
    int i;
    for (i=0;i<50;i++) Debug_Ram[i] = 0;
/* Debug stop */

    init_regs();
    while(1)
    {
    if ((Cmd_Bits & CmdPending) != 0)   /* Command Ready ? */
       {
       ExecuteCmd();
       }
    if ((Cmd_Bits & Tach_Bit) != 0)   /* If a Rotation happened */
```

```
        {
        if (TachUpLimit != 0)
        {
           if (Tach_Time > TachUpLimit)
           {
           asm("     SETC    INTM");  /* Disable intr while changing image */
           Ctrl_Image |= DSP_Intr:    /* Set DSP Int. Interupt the 188 */
           Ctrl_Port = Ctrl_Image:    /* Write to the port */
           Stat_Buffer[0] |= SpindleError: /* Set the Spindle Error Bit */
           asm("     CLRC    INTM");  /* Re enable interrupts */
           }
        }
        if (TachLowLimit != 0)
        {
           if (Tach_Time < TachLowLimit)
           {
           asm("     SETC    INTM");  /* Disable intr while changing image */
           Ctrl_Image |= DSP_Intr:    /* Set DSP Int. Interupt the 188 */
           Ctrl_Port = Ctrl_Image:    /* Write to the port */
           Stat_Buffer[0] |= SpindleError: /* Set the Spindle Error Bit */
           asm("     CLRC    INTM");  /* Re enable interrupts */
           }
        }
        if ((Stat_Buffer[0] & FineLoop) != 0)  /* If Fine Loop is closed */
           {
           if ((Stat_Buffer[0] & Jumpback_Out) != 0) /* do jumpback? */
              {
              Do_Jumpback(Seek_Out):
              }
           if ((Stat_Buffer[0] & Jumpback_In) != 0) /* do jumpback? */
              {
              Do_Jumpback(Seek_In):
              }
           }
        }
     if ((Ctrl_Image & LaserEnable) == LaserEnable)  /* Regulate laser power */
        {
```

```
            if ((Cmd_Bits & SenseSample) != 0)   /* Fwd Sense Sample Available */
               {
               RegulateLaser(LS);
               }
            }
         }
      }

      /*************************************************************************/
      /*  ExecuteCmd decodes and executes interupt recieved commands. when   */
      /*  the command is completed. this routine returns and enables command */
      /*  interupts.                                                          */
      /*************************************************************************/
      void ExecuteCmd (void)
      {
         int temp;

               /* Assume the command is understood and has a good check sum */
         Stat_Buffer[0] |= CmdComplete;   /* Set Command Complete */
         Stat_Buffer[0] &= ~BadChkSum;    /* Clear the Bad Check Sum bit */
         Stat_Buffer[0] &= ~UnknownCmd;   /* Clear the Unknown Command bit */

         if (CheckSum() == 0x00ff)  /* Test 1's Complement Check Sum on Commands */
         {
            temp = (CMD_Buffer[0] >> 8) & 0x00ff;
            switch (temp)
            {
              case 0 : SendStatus();     /* Command 0 = Status only */
                       break;
              case 1 : InitDrive();      /* Command 1 = Initialize drive */
                       break;
              case 2 : LaserOn();        /* Command 2 = Initialize the laser */
                       break;
              case 3:  CaptureFocus();   /* Command 3 = Capture Focus */
                       break;
              case 4:  CaptureFine();    /* Command 4 = Capture Fine Tracking */
                       break;
```

```
            case 5:  CaptureCoarse();  /* Command 5 = Close the Coarse Loop */
                     break;
            case 6:  ClosePinning();   /* Command 6 = Close the Pinning Loop */
                     break;
            case 7:  EnJumpbackIn();   /* Command 7 = Enable Jumpback In */
                     break;
            case 8:  EnJumpbackOut();  /* Command 8 = Enable Jumpback Out */
                     break;
            case 9:  DisJumpback();    /* Command 9 = Disable Jumpbacks */
                     break;
            case 0x0A: MultiTrackSeek(CMD_Buffer[1].Seek_In);
                     break;            /* Command A = Do a Seek toward spindle */
            case 0x0B: MultiTrackSeek(CMD_Buffer[1].Seek_Out);
                     break;            /* Command B = Do Seek away from spindle */
            case 0x0C: OpenLoops();    /* Command C = Open various loops */
                     break;
            case 0x0D: ClearDSPIntr(); /* Command D = Clear DSP to 188 Interrupt */
                     break;
            case 0x0E: VelTabStart (); /* Command E = Read start of Vel Table */
                     break;
            case 0x0F: ReadTimeTick(); /* Command F = Read 32 bit 20us Clock */
                     break;
            case 0x10: SetTachLimit(); /* Command 10h = Set Tach Pulse Limits */
                     break;
            case 0x80: ReadCodeRev();  /* Command 80h = Read DSP Code Rev */
                     break;
            case 0x81: ReadMemory();   /* Command 81h = Read DSP Ram Memory */
                     break;
            case 0x82: WriteMemory();  /* Command 82h = Write DSP Ram Memory */
                     break;
            default: BadCommand();     /* A bad command was sent*/
                     break;
          }
        }
        else BadCheckSum();
        asm("     SETC  INTM");       /* disable interupts */
        Cmd_Bits &= ~CmdPending;      /* Clear the Command Pending Bit */
```

```
    Cmd_Buff_Point = &Stat_Buffer[0];  /* Point to start of the status */
    Cmd_Bits |= 0x0004;                /* Set for the MSByte of the status */
    asm("     CLRC   INTM");           /* enable interupts */
    asm("     lamm   IMR");            /* Get the Interupt Mask Register */
    asm("     or     #0100h");         /* Set enable interupt 4 bit. bit 9 */
    asm("     samm   IMR");            /* Enable interupt 4 */
    asm("     intr   9");              /* do an interupt 4 */
}

void SendStatus (void)                 /* Command 0 = Status only */
{                              /* Do Nothing. Set up else where. just return */
}
void InitDrive (void)                  /* Command 1 = Init laser and servos */
{
    if ((Cmd_Buff_Point - &CMD_Buffer[0]) != 2)
    {
        BadCommand();
    }
    else
    {
        LaserOn();                     /* Turn the laser on */
        ClosePinning();                /* Close the pinning loop. enable Fine PA */
        Retract();                     /* Move the carriage to inner crash stop */
        CaptureFocus();                /* Capture Focus */
        CaptureCoarse();               /* Capture Coarse Tracking */
        Stat_Buffer[0] &= ~PinningLoop; /* open the Pinning loop */
        MultiTrackSeek(3000,Seek_Out);  /* Seek out to the disk */
        Delay(100);
        MultiTrackSeek(200,Seek_Out);  /* Seek out to the disk */
        FocusOffset();                 /* Find Optimum Focus for RPP */
        TrackOffset();                 /* Find RPP Center value */
    }
}
void LaserOn (void)                    /* Command 2 = laser on and calibrate */
{
    int TimeOut;
```

```
if ((Cmd_Buff_Point - &CMD_Buffer[0]) != 2)
{
    BadCommand();
}
else
{
    Stat_Buffer[0] &= LoopsOpen;  /* Clear Laser, Fine,Crs,Focus,Pin */
    Read_Sense = CMD_Buffer[1];   /* Save desired read sense value */
    Write_Sense = CMD_Buffer[2];  /* Save desired write sense value */
    ReadMSImage = 0;         /* Zero all the initial values */
    ReadMS_DAC = ReadMSImage;
    ReadLSImage = 0x4000;    /* Center range, 15 bits, initial value */
    ReadLS_DAC = (ReadLSImage << 1);  /* write out the value */
    WriteDacImage = 0;
    Write_DAC = WriteDacImage;
    asm("     SETC    INTM");  /* Disable intr while changing image */
    Ctrl_Image &= 0x0003;      /* Open loops, Laser off, leave DSP Int */
    Ctrl_Port = Ctrl_Image;    /* Write to the port */
    asm("     CLRC    INTM");  /* Re enable interrupts */
    Delay(100);    /* Delay 100 * 20 us (2000 us), let laser turn off */
    FwdSen_Zero = Int_FwdSen;  /* Save the laser off value */
    QSum_Zero = Int_QSum;      /* Save the laser off value */
    Fine_Zero = Int_Fine;      /* Save the laser off value */
    Focus_Zero = Int_Focus;    /* Save the laser off value */
    Crs_Zero = Int_Crs;        /* Save the laser off value */
    Stat_Buffer[0] &= ~LaserError;  /* Clr Laser Read Power Error */
    Stat_Buffer[0] |= LaserEnabled;  /* Laser Read Power is Not okay */
    asm("     SETC    INTM");  /* Disable intr while changing image */
    Ctrl_Image |= LaserEnable;   /* Set the laser read on bit */
    Ctrl_Port = Ctrl_Image; /* Turn on the laser */
    asm("     CLRC    INTM"); /* Re enable interrupts */
    TimeOut = 0;
    while(TimeOut < 2000)            /* Regulate the laser for 40ms */
      if ((Cmd_Bits & SenseSample) != 0) /* Fwd Sense Sample Available */
      {
      RegulateLaser(MS);
      TimeOut++;
```

```
        }
    }
}
/*************************************************************************/
/*   Capture Focus                                                       */
/*************************************************************************/
void CaptureFocus(void)         /* Command 3 = Capture Focus and Close Loop */
{
   int MaxQuadSum;
   int Counter;
   int TimeOut;

   Stat_Buffer[0] &= ~FocusLoop;     /* open the focus loop */
   MaxQuadSum = QSum_Zero;          /* Initialize the Quad Sum Value */
   FocMS_Image = 0;                 /* Set to zero value. signed integer */
   Foc_MS_DAC =  ZeroOffset;        /* Write zero value to the DAC */
   FocLS_Image = FocusStart;        /* Set Focus Current DAC to sweep start */
   Foc_LS_DAC = FocLS_Image + ZeroOffset;  /* Write DAC with zero offset */
   asm("     SETC    INTM");  /* Disable intr while changing image */
   Ctrl_Image |= FocusEnable;       /* Set the Focus PA Enable bit on */
   Ctrl_Port = Ctrl_Image;          /* Write out the port value */
   asm("     CLRC    INTM"); /* Re enable interrupts */
   Delay(150);                      /* Delay 3 ms (20 * 150us) */
               /* Find the Max Quad Sum Point */
   while (FocLS_Image > FocusStop)
   {
     Delay(5);                      /* Wait 100 us */
     if (Int_QSum > MaxQuadSum)     /* Get the max quad sum */
     {
        MaxQuadSum = Int_QSum;
     }
     Foc_LS_DAC = FocLS_Image + ZeroOffset;  /* Write DAC with zero offset */
     FocLS_Image -= 256;
   }
     /* Get the Quad Sum Threshold Level. 1/2 of Max Quad Sum */
   MaxQuadSum = ((MaxQuadSum - QSum_Zero) >> 1) + QSum_Zero;
     /* Find the Focus Point and close the loop */
```

```
while ((FocLS_Image < FocusStart) &
                        ((Stat_Buffer[0] & FocusLoop) != FocusLoop))
{
  for(Counter = 0; Counter < 32 ; ++Counter)          /* wait n times */
  {
    Delay(1);
        /* when quad sum okay and Focus negative */
    if ((Int_QSum > MaxQuadSum) & ((Int_Focus - Focus_Zero) > 0))
    {
       Stat_Buffer[0] |= FocusLoop; /* close the focus loop */
    }
  }
  Foc_LS_DAC = FocLS_Image + ZeroOffset;  /* Write DAC with zero offset */
  FocLS_Image += 256;
}
if ((Stat_Buffer[0] & FocusLoop) == FocusLoop)  /* If focus is closed */
{
     TimeOut = 0;
     while(TimeOut < 500)            /* Regulate the laser for 10ms */
     {
       if ((Cmd_Bits & SenseSample) != 0) /* Fwd Sense Sample Available */
       {
         RegulateLaser(MS);
         if (ReadLSImage < 0x4000) /* Center range. 15 bits. initial value */
         {
           ReadLSImage = ReadLSImage + 128;  /* Increment lsb of DAC */
           ReadLS_DAC = (ReadLSImage << 1);  /* write out the value */
         }
         else
         {
           ReadLSImage = ReadLSImage - 128;  /* Decrement lsb of DAC */
           ReadLS_DAC = (ReadLSImage << 1);  /* write out the value */
         }
         TimeOut++;
       }
     }
}
```

```
}
/*************************************************************************/
/*   Capture Fine Tracking                                               */
/*************************************************************************/
void CaptureFine(void)          /* Command 4 = Close the tracking Loop */
{
   int Counter;

   while ((Counter < 2000) & ((Stat_Buffer[0] & FineLoop) != FineLoop))
   {
     Delay(1);
     Counter++;
         /* when Fine Error is close to zero */
     if (abs(Int_Fine - Fine_Zero) < 0x1000)
     {
        Stat_Buffer[0] |= FineLoop;    /* close the fine loop around zero */
     }
   }
   if (Counter < 2000)
   {
     asm("     SETC    INTM");  /* Disable intr while changing image */
     Ctrl_Image |= FineEnable;  /* Set the Fine PA Enable bit on */
     Ctrl_Port = Ctrl_Image;    /* Write out the port value */
     asm("     CLRC    INTM"); /* Re enable interrupts */
     Stat_Buffer[0] |= FineLoop;   /* close the fine loop */
   }
}
/*************************************************************************/
/*   Capture Coarse Tracking                                             */
/*************************************************************************/
void CaptureCoarse(void)        /* Command 5 = Capture Coarse Tracking */
{
     Delay(50);                 /* Let the fine loop settle 1 ms */
     asm("     SETC    INTM");  /* Disable intr while changing image */
     Ctrl_Image |= CrsEnable;   /* Set the Coarse PA Enable bit on */
     Ctrl_Port = Ctrl_Image;    /* Write out the port value */
     asm("     CLRC    INTM");  /* Re enable interrupts */
```

```
    Stat_Buffer[0] |= CoarseLoop: /* close the Coarse loop */
}
/*************************************************************************/
/*   Close the Pinning Loop                                              */
/*************************************************************************/
void ClosePinning(void)          /* Command 6 = Close the Pinning Loop */
{
    asm("     SETC    INTM");    /* Disable intr while changing image */
    Ctrl_Image |= FineEnable:    /* Set the Fine PA Enable bit on */
    Ctrl_Port = Ctrl_Image:      /* Write out the port value */
    asm("     CLRC    INTM");    /* Re enable interrupts */
    Stat_Buffer[0] |= PinningLoop: /* close the Pinning loop */
}
void EnJumpbackIn(void)   /* Command 7 = Enable Jumpback In */
{
   Stat_Buffer[0] |= Jumpback_In:    /* Enable Jumpback Toward the Spindle */
}
void EnJumpbackOut (void)  /* Command 8 = Enable Jumpback Out */
{
    Stat_Buffer[0] |= Jumpback_Out:    /* Enable Jumpback Away from Spindle */
}
void DisJumpback(void)    /* Command 9 = Disable Jumpbacks */
{
    Stat_Buffer[0] &= ~Jumpback_In:     /* Disable Jumpback Toward Spindle */
    Stat_Buffer[0] &= ~Jumpback_Out:    /* Disable Jumpback Away from Spindle */
}
       /* Execute the Open Loops command from the 188 */
/*************************************************************************/
/*       Execute the Open Loops command from the 188                     */
/*************************************************************************/
void OpenLoops(void)              /* Command C = Open Loops */
{
    asm("     SETC    INTM");  /* Disable intr while changing image */
    if(CMD_Buffer[1] & 0x0100)     /* If true disable the laser */
      {
      Ctrl_Image &= ~LaserEnable: /* Clr Laser Bits of control port image */
      Stat_Buffer[0] &= ~LaserError:  /* Clr Laser Power Error Bit */
```

```
        Stat_Buffer[0] &= ~LaserEnabled; /* Laser Read Power is Not okay */
        }
    if(CMD_Buffer[1] & 0x0200)      /* If true disable the focus loop */
        {
        Ctrl_Image &= ~FocusEnable; /* Clr Focus Bit of control port image */
        Stat_Buffer[0] &= ~FocusError; /* Clr Focus Loop Error bit */
        Stat_Buffer[0] &= ~FocusLoop;  /* Open the Focus Loop */
        }
    if(CMD_Buffer[1] & 0x0400)      /* If true disable the Coarse loop */
        {
        Ctrl_Image &= ~CrsEnable;   /* Clr Coarse Bit of control port image */
        Stat_Buffer[0] &= ~CoarseLoop; /* Open the Coarse Loop */
        }
    if(CMD_Buffer[1] & 0x0800)      /* If true disable the Fine loop */
        {
        Ctrl_Image &= ~FineEnable;  /* Clr Fine Bit of control port image */
        Stat_Buffer[0] &= ~TrackingError;  /* Clr Tracking Loop Error Bit */
        Stat_Buffer[0] &= ~FineLoop;   /* Open the Fine Loop */
        }
    if(CMD_Buffer[1] & 0x1000)      /* If true disable the Pinning loop */
        {
        Ctrl_Image &= ~FineEnable;   /* Clr Fine Bit of control port image */
        Stat_Buffer[0] &= ~PinningLoop; /* Open the Fine Loop */
        }
    if(CMD_Buffer[1] & 0x2000)      /* If true clear the spindle error bit */
        {
        Stat_Buffer[0] &= ~SpindleError; /* Clear the spindle error bit */
        }
    Ctrl_Port = Ctrl_Image;         /* Clear the actual bits */
    asm("     CLRC    INTM");      /* Re enable interrupts */
}
        /* Execute the Clear DSP Interrupt command from the 188 */
void ClearDSPIntr(void)             /* Command D = Clear DSP to 188 Interrupt */
{
    asm("     SETC    INTM"); /* Disable interrupts while changing image */
    Ctrl_Image &= ~DSP_Intr;  /* Clear the LSBit of the control port image */
    Ctrl_Port = Ctrl_Image;   /* Clear the actual bit */
```

```
    asm("     CLRC    INTM"); /* Re enable interrupts */
}
        /* Execute the Velocity Table Start Address */
void VelTabStart(void)   /* Command 0Eh = Read Velocity Table Start Address */
{
    Stat_Buffer[1] = ((int) &Vel_Table); /* Store Vel Table Start Addr */
    Stat_Buffer[2] = ((int) &InverseTime); /* Store Inverse Time Start Addr */
}
        /* Execute the Read Time Tick Counter */
void ReadTimeTick()        /* Command 0Fh = Read Time Tick Counter */
{
    asm("     SETC    INTM");  /* Disable intr while storing the values */
    Stat_Buffer[1] = Count_20_MSW; /* Store Time Tick MSWord */
    Stat_Buffer[2] = Count_20_LSW; /* Store Time Tick LSWord */
    asm("     CLRC    INTM"); /* Re enable interrupts */
}
        /* Execute the Set Tach time limits */
void SetTachLimit(void)      /* Command 10h = Set the tach pulse limits */
{
    TachUpLimit = CMD_Buffer[1];   /* Save the upper limit value */
    TachLowLimit = CMD_Buffer[2];  /* Save the lower limit value */
}
        /* Execute the Code Revision Read command from the 188 */
void ReadCodeRev(void)       /* Command 80h = Read the DSP Code Rev Level */
{
    static char Rev[5]={ Revision };  /* Rev level reported by ReadCodeRev */

    Stat_Buffer[1] = ((int) Rev[0] << 8);  /* Get the Character. SHL 8 */
    Stat_Buffer[1] |= (int) Rev[1];      /* Get 2nd Character in buffer */
    Stat_Buffer[2] = ((int) Rev[2] << 8);  /* Get the 3rd Character. SHL 8 */
    Stat_Buffer[2] |= (int) Rev[3];      /* Get 4th Character in buffer */
}
        /* Execute the Memory read command from the 188 */
void ReadMemory(void)         /* Command 81h = Read DSP Ram Memory */
{
    int *Temp_P1,*Temp_P2,Temp;
```

```
    Temp_P1 = &CMD_Buffer[0];           /* Point to 1st Cmd Byte */
    ++Temp_P1;                          /* Point to Address Value */
    Temp_P2 = (int *) *Temp_P1;         /* Load Address into Temp_P2 */
    Temp_P1 = &Stat_Buffer[0];          /* Point to the Status Buffer */
    ++Temp_P1;                          /* Point to the 1st data word */
    while (Temp_P1 < &CMD_Buffer[0])    /* Move all the words */
    {
       *Temp_P1 = *Temp_P2;             /* Store a word of data */
       ++Temp_P1;                       /* Point to next data word */
       ++Temp_P2;                       /* Point to next storage location */
    }
}
        /* Execute the Memory write command from the 188 */
void WriteMemory(void)       /* Command 81h = Write DSP Ram Memory */
{
    int *Temp_P1,*Temp_P2,Temp;

    Temp_P1 = &CMD_Buffer[0];           /* Point to 1st Cmd Byte */
    ++Temp_P1;                          /* Point to Address Value */
    Temp_P2 = (int *) *Temp_P1;         /* Load Address into Temp_P2 */
    ++Temp_P1;                          /* Point to 1st data word */
    while (Temp_P1 <= Cmd_Buff_Point)   /* Move all the words */
    {
       *Temp_P2 = *Temp_P1;             /* Store a word of data */
       ++Temp_P1;                       /* Point to next data word */
       ++Temp_P2;                       /* Point to next storage location */
    }
}
        /* Set up the status buffer saying the command was not understood */
void BadCommand(void)       /*Command ? = an undefined command */
{
    Stat_Buffer[0] &= ~CmdComplete;   /* Clear Command Complete */
    Stat_Buffer[0] |= UnknownCmd;     /* Set Unknown Command bit */
}
        /* Set up the status buffer saying the checksum was bad */
void BadCheckSum(void)
{
```

```
    Stat_Buffer[0] &= ~CmdComplete:   /* Clear Command Complete */
    Stat_Buffer[0] |= BadChkSum:      /* Set Bad Check Sum bit */
}

    /* Calculate the checksum on the command buffer */
int CheckSum(void)
{
    int *Temp_Point.sum:

    sum = 0:                                  /* Initialize the Sum Value */
    Temp_Point = &CMD_Buffer[0]:              /* Point to 1st Cmd Byte */
    while (Temp_Point <= Cmd_Buff_Point)      /* Do Summation for all bytes */
    {
       sum += *Temp_Point + (*Temp_Point >> 8):  /* Sum both bytes of word */
       ++Temp_Point:                          /* Point to next command word */
    }
    sum &= 0x00FF:                            /* Clear MSByte */
    return(sum):                              /* Return with Sum Value */
}
/*************************************************/
/*    Delay so many 20us time ticks then return   */
/*************************************************/
void Delay(int ticks)
{
    int Stop_Time:

    Stop_Time = Count_20_LSW + ticks: /* Calculate the stop time */
    while (Stop_Time != Count_20_LSW) /* Repeat until counter equals stop */
    {
    }
}
/*****************************************************************************/
/*     Regulate the laser read power                                         */
/*****************************************************************************/
void RegulateLaser(int MS_LS)
{     /* Integrate around Read_Sense. Desired -  Unsign ADC Value */
  int ReadPCLError:
```

```
    /* Int_FwdSen is a 10 bit unsigned (positive) number */
  ReadPCLError = ((Int_FwdSen - FwdSen_Zero) - Read_Sense);
  if (MS_LS == LS)
    {
    ReadLSImage = (ReadLSImage - ReadPCLError);
    if (ReadLSImage < 0)  /* Over or Under Flow */
      {
      if (ReadPCLError > 0)   /* UNDER FLOW */
        {
        ReadLSImage = 0;
        }
      else
        {
        ReadLSImage = 0x7FFF;
        }
      RegulateLaser(MS);   /* Regulate with the Large bit step */
      }
    ReadLS_DAC = (ReadLSImage << 1);
    }
  else
    {
    ReadMSImage = (ReadMSImage - ReadPCLError);
    if (ReadMSImage < 0)
      {
      if (ReadPCLError > 0)   /* UNDER FLOW */
        {
        ReadMSImage = 0;
        }
      else
        {
        ReadMSImage = 0x7FFF;
        }
      }
    ReadMS_DAC = (ReadMSImage << 1);
    }
  if (abs(ReadPCLError) < ReadPowerTol)
    {
```

```
        Stat_Buffer[0] &= ~LaserError;  /* Laser Read Power is Okay */
        }
        else
        {
        Stat_Buffer[0] |= LaserError;   /* Laser Read Power is Not okay */
        }
      asm("     SETC   INTM");      /* Disable interupts */
      Cmd_Bits &= ~SenseSample;     /* Clear the Sense Available Bit */
      asm("     CLRC   INTM");      /* enable interupts */
}
/*************************************************************************/
/*   Program Name      : Drive.h                                         */
/*   Description       : 4x 5.25" DSP Servo controller header file       */
/*   Part Number       : 562096                                          */
/*   Date              : 8/12/93                                         */
/*   O/S               : N/A                                             */
/*   Compiler          : TI TMS320C2x/C5x Compiler.#TMDS3242855-02.Rel. 6.0 */
/*   Support Packages  : N/A                                             */
/*   Author            : Dave Schell                                     */
/*   Required Files    : Drive.c.Interupt.asm.C50_init.asm.Seek.c.Drive.h */
/*                     : Recal.c                                         */
/*   Hardware Required : Part # XXXXXX                                   */
/*   Install. Instr.   : Link in with Drive code                         */
/*   Operating Instr.  : N/A                                             */
/*                                                                       */
/*   Rev History                                                         */
/*     Date     Rev  C#  Init     Change Description                     */
/*   4/14/94    XA   00   DLS   Initial Release                          */
/*************************************************************************/

enum Seek_Dir {Seek_Out,Seek_In};  /* direction const Jumpback and seek */
enum Laser_Reg {LS,MS};            /* LS,MS Test for laser regulation */

#define Revision "XA00"            /* The current Revision Level */

#define Retr_Accel 6345            /* (2.0 Gs / 0.0003152 Gs/bit) */
```

```
#define Retr_Pulses 100          /* Number of Retract Pulses */

#define Foc_Step 655             /* Focus Step Size for Auto Focus */

#define JB_Accel 9835            /* (3.1 Gs / 0.0003152 Gs/bit) */
#define JB_Decel -13959          /* (4.4 Gs / 0.0003152 Gs/bit) */
#define Seek_Accel 18718         /* (5.9 Gs / 0.0003152 Gs/bit) */
#define Seek_Decel -13959        /* (4.4 Gs / 0.0003152 Gs/bit) */
#define Hi_BW_Tracks 10          /* Seek length for using high Bandwidth */
#define HiSeekGain 149           /* 149, Wide BW (750 Hz) Gain constant */
#define LowSeekGain 99           /* 99, Low BW (500 Hz) Gain constant */

#define CmdComplete 0x8000       /* Status Word Command Complete Bit */
#define BadChkSum 0x4000         /* Status Word Bad Check Sum Bit */
#define UnknownCmd 0x2000        /* Status Word Unknown Command Bit */
#define TrackingError 0x1000     /* Status Word Tracking Error Bit */
#define SpareBit 0x0800          /* Status Word Spare Bit */
#define FocusError 0x0400        /* Status Word Focus Error Bit */
#define LaserError 0x0200        /* Status Word Laser Control Error Bit */
#define FocusLoop 0x0100         /* Status Word Focus Loop Closed Bit */
#define FineLoop 0x0080          /* Status Word Fine Loop Closed Bit */
#define CoarseLoop 0x0040        /* Status Word Coarse Loop Closed Bit */
#define PinningLoop 0x0020       /* Status Word Pinning Loop Closed Bit */
#define SpindleError 0x0010      /* Status Word Tach Out of Spec Bit */
#define LaserEnabled 0x0008      /* Status Word Laser Read Power is on */
#define Jumpback_In 0x0004       /* Status Word Jumping Back In Bit */
#define Jumpback_Out 0x0002      /* Status Word Jumping Back Out Bit */
#define Bad_Seek 0x0001          /* Status Word Bad Seek/no Vel Table Bit */
#define LoopsOpen 0xE00F         /* Clear Tach, Focus, Coarse, Fine, Pin */

#define SenseSample 0x0080       /* Sense Sample Available Bit */
#define CmdPending 0x0001        /* Command Bits, Command pending Bit */
#define Tach_Bit 0x0200          /* Command Bits, Motor Tach Bit */

#define ReadPowerTol 0x0005      /* Read Power Okay Tolerance */
#define FocusEnable  0x0080      /* Control Port Focus PA Enable Bit */
```

```
#define FineEnable    0x0040        /* Control Port Fine PA Enable Bit */
#define CrsEnable     0x0020        /* Control Port Fine PA Enable Bit */
#define DTCS_Clk_Pol  0x0010        /* Track Crossing Clock Polarity */
#define LaserEnable   0x0008        /* Control Port Laser Enable Bit */
#define Software_TP   0x0002        /* Control Port Software Test Point Bit */
#define DSP_Intr      0x0001        /* Control Port DSP to 188 interrupt Bit */

#define FocusStart  32000           /* Focus Capture Sweep Current Minimum */
#define FocusStop  -32000           /* Focus Capture Sweep Current Maximum */
#define ZeroOffset 0x8000           /* Zero Offset value for the DACs */
#define MaxPositive 0x7FFF          /* Max positive signed integer value */
#define MaxNegative 0x8000          /* Max negative signed integer value */

extern int Read_Sense:      /* Laser Read Sense Desired Level */
extern int Write_Sense:     /* Laser Write Sense Desired Level */
extern int ReadMSImage:     /* Laser Read DAC Bit Image */
extern int ReadLSImage:     /* Laser Read DAC Bit Image */
extern int WriteDacImage:   /* Laser Write DAC 16 Bit Image */
extern int MaxRPP:          /* Maximum RPP Value seen during a jumpback */
extern int MinRPP:          /* Minimum RPP Value seen during a jumpback */


extern int FwdSen_Zero:     /* ADC Forward Sense Value with the laser off */
extern int QSum_Zero:       /* ADC Quad Sum Value with the laser off */
extern int Fine_Zero:       /* ADC Tracking Value with the laser off */
extern int Focus_Zero:      /* ADC Focus Value with the laser off */
extern int Crs_Zero:        /* ADC Coarse Error Zero */
extern int FineDacZero:     /* Fine DAC Zero or Seek Accel Value */
extern int CrsDacZero:      /* Crs DAC Zero or Seek Accel Value */
extern int FocLS_Image:     /* Focus Current LS (Capture) DAC 16 Bit Image */
extern int FocMS_Image:     /* Focus Current MS (Servo) DAC 16 Bit Image */
extern int FineDacImage:    /* Fine Current (Servo) DAC 16 Bit Image */
extern int Cmd_Bits:        /* Command ready Status Flag */
extern int *Cmd_Buff_Point: /* Command Buffer Pointer */
extern int Stat_Buffer[5]:  /* The first word of the status buffer */
extern int CMD_Buffer[10]:  /* The first word of the command buffer */
extern int SPC:             /* The Serial Control Input Port */
```

```
extern int Track_Cnt;        /* The Track Crossing Counter Input Port */
extern int Ctrl_Port;        /* The Output Control Port */
extern int Ctrl_Image;       /* The memory image of the Control Port */
extern int Foc_LS_DAC;       /* Focus DAC Memory Location */
extern int Foc_MS_DAC;       /* Focus DAC Memory Location */
extern int Foc_Err_Cnt;      /* Focus Out of limit sample counter */
extern int Focus_Limit;      /* Focus Error Spec Limit Value */
extern int Fine_DAC;         /* Fine DAC Memory Location */
extern int Fine_Err_Cnt;     /* Focus Out of limit sample counter */
extern int Fine_Limit;       /* Focus Error Spec Limit Value */
extern int Crs_DAC;          /* Coarse DAC Memory Location */
extern int Write_DAC;        /* Write DAC Memory Location */
extern int ReadLS_DAC;       /* Read DAC Memory Location */
extern int ReadMS_DAC;       /* Read DAC Memory Location */
extern int Spare_DAC;        /* Spindle or Test DAC Memory Location */
extern int Int_Focus;        /* Focus Interupt Value */
extern int Int_Fine;         /* Fine Interupt Value */
extern int Int_Crs;          /* Coarse Interupt Value */
extern int Int_FwdSen;       /* Forward Sense Interupt Value */
extern int Int_QSum;         /* Quad Sum Interupt Value */
extern int Int_Test;         /* Test Interupt Value */
extern int Count_20_MSW;     /* 20us Time tick counter. upper 16 bit value */
extern int Count_20_LSW;     /* 20us Time tick counter. lower 16 bit value */
extern int TachUpLimit;      /* Tach Pulse upper time limit */
extern int TachLowLimit;     /* Tach Pulse lower time limit */
extern int Tach_Time;        /* Tach Pulse Recurrence Interval time  */
extern int Vel_Table[384];   /* Seek Velocity Table Starting Address */
extern int InverseTime[25];  /* Inverse Time Table for seek velocity calcs */
extern int Debug_Ram[50];    /* Inverse Time Table for seek velocity calcs */

void init_regs(void);        /* Initialize the DSP registars */
void ExecuteCmd(void);
void SendStatus(void);
void InitDrive(void);
void LaserOn(void);
void CaptureFocus(void);
void CaptureFine(void);
```

```
void CaptureCoarse(void);
void ClosePinning(void);
void EnJumpbackIn(void);
void EnJumpbackOut(void);
void DisJumpback(void);
void OpenLoops(void);
void ClearDSPIntr(void);
void VelTabStart(void);
void ReadTimeTick(void);
void SetTachLimit(void);
void ReadCodeRev(void);
void ReadMemory(void);
void WriteMemory(void);
void BadCommand(void);
void BadCheckSum(void);
int CheckSum(void);
void Delay(int ticks);
void RegulateLaser(int);
void Do_Jumpback(int);
void Track_Capture(int,int);
void MultiTrackSeek(int,int);
void Retract(void);          /* Move the carriage to inner crash stop */
void FocusOffset(void);      /* Find the Zero Offsets for Max RPP */
void TrackOffset(void);      /* Find Zero Offset at (Max Rpp + Min RPP)/2 */
int FindPeaktoPeak(void);    /* Find the peak to peak RPP Value */
/*****************************************************************************/
/*   Program Name      : Seek.c                                              */
/*   Description       : 4x 5.25" DSP Servo controller seek routines         */
/*   Part Number       : 562096                                              */
/*   Date              : 12/12/93                                            */
/*   O/S               : N/A                                                 */
/*   Compiler          : TI TMS320C2x/C5x Compiler.#TMDS3242855-02.Rel. 6.0 */
/*   Support Packages  : N/A                                                 */
/*   Author            : Dave Schell                                         */
/*   Required Files    : Drive.c.Interupt.asm.C50_init.asm.Seek.c.Drive.h    */
/*                     : Recal.c                                             */
/*   Hardware Required : Part # XXXXXX                                       */
```

```
/*   Install. Instr.    : Link in with Drive code                          */
/*   Operating Instr.   : N/A                                              */
/*                                                                         */
/*   Rev History                                                           */
/*     Date      Rev  C#  Init      Change Description                     */
/*   04/14/94    XA   00   DLS    Initial Release                          */
/*************************************************************************/

#include "drive.h"

/*************************************************************************/
/*   Multi Track Seek in or out                                          */
/*************************************************************************/
void MultiTrackSeek(Tracks.In_Out)
int Tracks.In_Out:
{
  int Accel.Decel.Sign.i.DeltaTime.OldTime.NewTime:
  int MeasuredVel.DesiredVel.TrackCount.Old_Trk_Cnt.New_Trk_Cnt.DeltaTracks:
  int LoopError.LoopGain.VelError.MaxVelError.OldCrsDacZero:
  int OldHalfTrack.HalfTrack:

  TrackCount = Tracks:                  /* Get the number of track to seek */
       /* Setup the direction dependent parameters */
  if (In_Out == Seek_In)
  {
     Sign = 1:
     asm("     SETC    INTM"):   /* Disable intr while changing image */
/* debug */
     Ctrl_Port = Ctrl_Image |= Software_TP:    /* Write to the port */
     Ctrl_Port = Ctrl_Image &= ~Software_TP:    /* Write to the port */
/* debug */
     Ctrl_Image |= DTCS_Clk_Pol: /* Set the DTCS Clock Polarity Bit */
     Ctrl_Port = Ctrl_Image:    /* Write to the port */
     asm("     CLRC    INTM"):  /* Re enable interrupts */
  }
  else
  {
```

```
        Sign = -1;
        asm("     SETC    INTM");    /* Disable intr while changing image */
    /* debug */
        Ctrl_Port = Ctrl_Image |= Software_TP;    /* Write to the port */
        Ctrl_Port = Ctrl_Image &= ~Software_TP;    /* Write to the port */
    /* debug */
        Ctrl_Image &= ~DTCS_Clk_Pol; /* Clear the DTCS Clock Polarity Bit */
        Ctrl_Port = Ctrl_Image;      /* Write to the port */
        asm("     CLRC    INTM");    /* Re enable interrupts */
    }
    Decel = Seek_Decel * Sign;


            /* If velocity table initilaized and focus is closed */
    if ((Vel_Table[0] != -1) && (Stat_Buffer[0] & FocusLoop) && (TrackCount > 0)
        && (InverseTime[0] != -1))
    {                                       /* then do the seek */
      Stat_Buffer[0] &= ~Bad_Seek;          /* Clear the bad seek status bit */
      if (TrackCount < 5)
      {
         for (i = 0; i < TrackCount; i++) Do_Jumpback(In_Out);
      }
      else
      {
         if (TrackCount > Hi_BW_Tracks)
         {
           Accel = MaxPositive * Sign;
           LoopGain = HiSeekGain * Sign;   /* Set the Seek Gain for High BW */
         }
         else
         {
           Accel = Seek_Accel * Sign;
           LoopGain = LowSeekGain * Sign;  /* Set the Seek Gain for low BW */
         }
         MaxVelError = abs(MaxPositive / LoopGain);
         OldTime = Count_20_LSW;    /* Get the start time */
         MeasuredVel = 0;           /* Starting Velocity = 0 */
```

```
Stat_Buffer[0] &= ~FineLoop; /* open the fine loop */
FineDacImage = Accel;      /* Update the Image Value */
Fine_DAC = (Accel + ZeroOffset); /* write out the acceleration value */
OldCrsDacZero = CrsDacZero;
CrsDacZero = Accel;        /* Output the coarse loop accel value */
Delay(5);                  /* Wait 100 us for RPP to go away from zero */
New_Trk_Cnt = (Track_Cnt & 0x00FF); /* Get the track counter value */
Old_Trk_Cnt = New_Trk_Cnt; /* Set them equal before starting loop */
if ((Old_Trk_Cnt & 0x0080) == 0) HalfTrack = 1; else HalfTrack = 0;
OldHalfTrack = HalfTrack;  /* Set them equal before starting loop */
while (TrackCount > 1)
{
   if (MeasuredVel < MaxPositive )  /* Velocity > 80.0 mm/s */
   {
     i = 0;
     while ((i < 6000) & (New_Trk_Cnt==Old_Trk_Cnt))
     {
       i++;
       New_Trk_Cnt = (Track_Cnt & 0x00FF); /* Get track count value */
     }
   }
   else
   {
     Delay(1);  /* at high velocity delay 1 time tick */
   }
   New_Trk_Cnt = (Track_Cnt & 0x00FF); /* Get track count value */
   NewTime = Count_20_LSW;      /* Get the current time */
   if ((New_Trk_Cnt & 0x0080) == 0) HalfTrack = 1; else HalfTrack = 0;
   DeltaTracks = (New_Trk_Cnt - Old_Trk_Cnt) & 0x007F;  /* Get Delta */
   Old_Trk_Cnt = New_Trk_Cnt;   /* Save the Track Count Value */
   TrackCount -= DeltaTracks;   /* Update the track counter */
               /* Calculate the number of half tracks */
   DeltaTracks = (DeltaTracks * 2) + HalfTrack - OldHalfTrack;
   OldHalfTrack = HalfTrack;
   DeltaTime = (NewTime - OldTime) - 1;
   OldTime = NewTime;           /* Save the time for next time */
   if (DeltaTime > 24) DeltaTime = 24;
```

```
MeasuredVel = DeltaTracks * InverseTime[DeltaTime];
if (TrackCount > 9280)      /* Tracks > 128+(8*128)+(64*128) */
{
   DesiredVel = Vel_Table[383];
}
else if (TrackCount > 1152)
{
   DesiredVel = Vel_Table[256 + ((TrackCount - 1152) >> 6)];
}
else if (TrackCount > 128)
{
   DesiredVel = Vel_Table[128 + ((TrackCount - 128) >> 3)];
}
else
{
   DesiredVel = Vel_Table[TrackCount];
}
/* Loop Error = LoopGain * (Desired Velocity - Measured Velocity) */
VelError = (DesiredVel - MeasuredVel);
if (abs(VelError) < MaxVelError)
{
  LoopError = LoopGain * VelError;
}
else
{
  if (VelError > 0)
  {
    LoopError = (MaxPositive * Sign);
  }
  else
  {
    LoopError = - (MaxPositive * Sign);
  }
}
FineDacImage = LoopError;     /* Update the Image Value */
Fine_DAC = (LoopError+ZeroOffset);    /* write out accel value */
CrsDacZero = LoopError;  /* Output the coarse loop accel value */
```

```
            }
            CrsDacZero = OldCrsDacZero;
            Track_Capture(In_Out.Decel);
          }
      }
      else                                  /* If Vel Table Not initialized */
      {
        Stat_Buffer[0] |= Bad_Seek;         /* Set bad seek status bit */
      }
  }
  /*****************************************************************************/
  /*     Do a Jumpback in or out                                               */
  /*****************************************************************************/
  void Do_Jumpback(In_Out)
  int In_Out;
  {
      int Accel,Decel,i;

      MaxRPP = 0;
      MinRPP = 0;          /* Initialize the min and max values to 0 */
      asm("     SETC   INTM");     /* Disable interupts */
      Cmd_Bits &= ~Tach_Bit;       /* Clear the Tach Bit */
      asm("     CLRC   INTM");     /* enable interupts */
      if (In_Out == Seek_In)       /* do jumpback in */
         {
         Accel = JB_Accel;
         Decel = JB_Decel;
         }
      else                         /* do jumpback out */
         {
         Accel = -JB_Accel;
         Decel = -JB_Decel;
         }
      Stat_Buffer[0] &= ~FineLoop; /* open the fine loop */
      FineDacImage = Accel;     /* Update the Image Value */
      Fine_DAC = (Accel + ZeroOffset);   /* write out the acceleration value */
      for (i = 0; i < 6; i++)    /* Wait 120 us for RPP to go away from zero */
```

```
        {
           Delay(1);
           if (MaxRPP < Int_Fine) MaxRPP = Int_Fine; /* Find RPP Min and Max */
           if (MinRPP > Int_Fine) MinRPP = Int_Fine;
        }
        Track_Capture(In_Out,Decel);
    }
    /*************************************************************************/
    /*     Track Capture After Jumpback or a seek                            */
    /*************************************************************************/
    void Track_Capture (In_Out,Decel)
    int In_Out,Decel;
    {
        int Counter,i;
        int OldDacZero;

                    /* Get ready for Capture */
        OldDacZero = FineDacZero;
        FineDacZero = Decel;
        Counter = 0;
                /* Wait for 1/2 track */
        if (In_Out == Seek_In)  /* Wait for RPP to go low */
           {
           while (((Int_Fine - Fine_Zero) < 0) & (Counter < 6000))
             {
             Counter++;
             if (MinRPP > Int_Fine) MinRPP = Int_Fine; /* Find RPP Min */
             }
           }
        else                    /* wait for RPP to go high */
           {
           while (((Int_Fine - Fine_Zero) > 0) & (Counter < 6000))
             {
             Counter++;
             if (MaxRPP < Int_Fine) MaxRPP = Int_Fine;  /* Find RPP Max */
             }
           }
```

```
        FineDacImage = Decel;      /* Update the Image Value */
        Fine_DAC = (Decel + ZeroOffset);  /* Write out the deceleration value */
        for (i = 0; i < 3; i++)    /* Wait 60 us */
        {
          Delay(1);
          if (MaxRPP < Int_Fine) MaxRPP = Int_Fine; /* Find RPP Min and Max */
          if (MinRPP > Int_Fine) MinRPP = Int_Fine;
        }
        Stat_Buffer[0] |= FineLoop;  /* Close the fine tracking loop */
        for (i = 0; i < 3; i++)    /* Wait 80 us */
        {
          Delay(1);
          if (MaxRPP < Int_Fine) MaxRPP = Int_Fine; /* Find RPP Min and Max */
          if (MinRPP > Int_Fine) MinRPP = Int_Fine;
        }
        FineDacZero = OldDacZero; /* Remove the deceleration pulse */
    }
    /*************************************************************************/
    /*     Retract the Carriage to the inner Crash Stop                      */
    /*************************************************************************/
    void Retract (void)
    {
        int i;

        asm("     SETC    INTM");  /* Disable intr while changing image */
        Ctrl_Image |= CrsEnable;   /* Set the Coarse PA Enable bit on */
        Ctrl_Port = Ctrl_Image;    /* Write out the port value */
        asm("     CLRC    INTM");  /* Re enable interrupts */
        for (i = 0; i < Retr_Pulses; i++)
        {
          Crs_DAC = Retr_Accel + ZeroOffset;   /* Accel toward the spindle */
          Delay(250);              /* for 5 ms */
          Crs_DAC = ZeroOffset;    /* Coast toward the spindle */
          Delay(750);              /* for 15 ms */
        }
        Crs_DAC = Retr_Accel + ZeroOffset;   /* Hold at inner crash stop */
        Delay(5000);               /* for 100 ms */
```

```
}
/*****************************************************************************/
/*   Program Name      : Recal.c                                               */
/*   Description       : 4x 5.25" DSP Servo controller recalibration stuff     */
/*   Part Number       : 562096                                                */
/*   Date              : 2/2/94                                                */
/*   O/S               : N/A                                                   */
/*   Compiler          : TI TMS320C2x/C5x Compiler,#TMDS3242855-02,Rel. 6.0    */
/*   Support Packages  : N/A                                                   */
/*   Author            : Dave Schell                                           */
/*   Required Files    : Drive.c,Interupt.asm,C50_init.asm,Seek.c,Drive.h      */
/*                     : Recal.c                                               */
/*   Hardware Required : Part # XXXXXX                                         */
/*   Install. Instr.   : Link in with Drive code                               */
/*   Operating Instr.  : N/A                                                   */
/*                                                                             */
/*   Rev History                                                               */
/*     Date     Rev  C#  Init     Change Description                         */
/*    4/14/94   XA   00   DLS    Initial Release                              */
/*****************************************************************************/

#include "drive.h"

/*****************************************************************************/
/*   Set focus zero to the optimum RPP Focus Offset                          */
/*****************************************************************************/
void FocusOffset(void)     /* Find the Zero Offsets for Max RPP */
{
    int j,PeaktoPeak[3],Center;

    for (j = 0; j < 3; j++) PeaktoPeak[j] = 0;
    Center = Focus_Zero;
    for (j = 0; j < 3; j++)
    {
      Focus_Zero = Center - Foc_Step + (j * Foc_Step);
      PeaktoPeak[j] = FindPeaktoPeak();
    }
```

```
        if (PeaktoPeak[0] > PeaktoPeak[2])
        {
          while ((j < 20) & (PeaktoPeak[0] > PeaktoPeak[2]))
          {
            Center -= Foc_Step;
            Focus_Zero = Center - Foc_Step;
            PeaktoPeak[2] = PeaktoPeak[1];
            PeaktoPeak[1] = PeaktoPeak[0];
            PeaktoPeak[0] = FindPeaktoPeak();
          }
        }
        else
        {
          while ((j < 20) & (PeaktoPeak[2] > PeaktoPeak[0]))
          {
            Center += Foc_Step;
            Focus_Zero = Center + Foc_Step;
            PeaktoPeak[0] = PeaktoPeak[1];
            PeaktoPeak[1] = PeaktoPeak[2];
            PeaktoPeak[2] = FindPeaktoPeak();
          }
        }
        Focus_Zero = Center;
    }
    /*************************************************************************/
    /*   Set RPP Zero to the center of the peak to peak RPP                  */
    /*************************************************************************/
    void TrackOffset(void)     /* Find Zero Offset at (Max Rpp + Min RPP)/2 */
    {
      int i,Max,Min;

      Max = Min = 0;        /* Initialize the value */

      for (i = 0; i < 8; i++)
      {
        Delay(50);
        Do_Jumpback(Seek_In);
```

```
        Max += MaxRPP >> 4;  /* Max Value / 16 */
        Min += MinRPP >> 4;  /* Min Value / 16 */
      }
      for (i = 0; i < 8; i++)
      {
        Delay(50);
        Do_Jumpback(Seek_Out);
        Max += MaxRPP >> 4;  /* Max Value / 16 */
        Min += MinRPP >> 4;  /* Min Value / 16 */
      }
      Fine_Zero = (Max + Min) >> 1;   /* (Max RPP + Min RPP) / 2 */
    }
    /*****************************************************************************/
    /*   Measure the peak to peak RPP value by doing jumpbacks in and out       */
    /*****************************************************************************/
    int FindPeaktoPeak(void)  /* Find the peak to peak RPP Value */
    {
      int i,PeaktoPeak;

      PeaktoPeak = 0;        /* Initialize the value */

                  /* Clear the Tach Bit */
      asm("     SETC   INTM");     /* Disable interupts */
      Cmd_Bits &= ~Tach_Bit;       /* Clear the Tach Bit */
      asm("     CLRC   INTM");     /* enable interupts */

                  /* Wait for the tach to go high */
      while (((Cmd_Bits & Tach_Bit) == 0) && (i++ < 2500)) Delay(1);

      for (i = 0; i < 8; i++)
      {
        Delay(50);
        Do_Jumpback(Seek_In);
        PeaktoPeak += ((MaxRPP - MinRPP) >> 5);  /* Peak to peak / 32 */
      }
      for (i = 0; i < 8; i++)
      {
```

```
        Delay(50);
        Do_Jumpback(Seek_Out);
        PeaktoPeak += ((MaxRPP - MinRPP) >> 5);  /* Peak to peak / 32 */
      }
    return(PeaktoPeak);  /* Return the average pk-pk value/2 */
}
;*****************************************************************************
;   Program Name      : Interupt.asm
;   Description       : DSP Interupt handling routines for the 4x 5.25"
;   Part Number       : 562096
;   Date              : 8/12/93
;   O/S               : N/A
;   Compiler          : TI TMS320C2x/C5x Compiler. #TMDS3242855-02.Rel. 6.0
;   Support Packages  : N/A
;   Author            : Dave Schell
;   Required Files    : Drive.c.Interupt.asm.C50_init.asm.Seek.c.Drive.h
;                     : Recal.c
;   Hardware Required : Part # XXXXXX
;   Install. Instr.   : Link in with Drive code
;   Operating Instr.  : N/A
;
;   Rev History
;     Date     Rev  C#  Init     Change Description
;   4/14/94    XA   00   DLS     Initial Release
;*****************************************************************************

        .title  "Processor Interupt Handlers"

        .length 60

        .mmregs

      .def    ISR1,ISR2,CMD_Intr,Timer
      .def    Tach,Old_Tach_Time,_Tach_Time
      .def    _TachUpLimit,_TachLowLimit
      .def    RCV,XMT,TDMRCV,TDMXMT,TRP,NMISR
      .def    _Count_20_LSW,_Count_20_MSW,_Cmd_Bits,_CMD_Buffer
```

```
        .def    _Fine_DAC,_Crs_DAC,_ReadLS_DAC,_ReadMS_DAC,_Write_DAC
        .def    _Foc_LS_DAC,_Foc_MS_DAC,_Spare_DAC,_Track_Cnt,_SPC
        .def    _Int_QSum,_Int_Fine,_Int_FwdSen,_Int_Crs,_Int_Focus,_Int_Test
        .def    _Stat_Buffer,_Cmd_Buff_Point,_Ctrl_Image,_Ctrl_Port,Sign_Bit
        .def    _QSum_Zero,_Fine_Zero,_Focus_Zero,_FwdSen_Zero,_Crs_Zero
        .def    _FocMS_Image,_FocLS_Image,_FineDacImage,_CrsDacImage
        .def    Focus_N1,Focus_N2,Focus_N3,Focus_D2,Focus_D3,Focus_G
        .def    Fine_N1,Fine_N2,Fine_N3,Fine_D2,Fine_D3,Fine_G
        .def    Crs_N1,Crs_N2,Crs_N3,Crs_D2,Crs_D3,Crs_G,Pin_G
        .def    _Debug_Ram,Focus_Error,Old_Focus_1,_FineDacZero,_CrsDacZero
        .def    _Foc_Err_Cnt,_Focus_Limit,_Fine_Err_Cnt,_Fine_Limit


;  I/O Definition
_SPC        .set    00022h      ;Serial Port Control Register
Cmd_Port    .set    00050h      ;80188/DSP Communication Port
_Ctrl_Port  .set    00051h      ;I/O Control Port, laser, Power Amps, etc.
_Track_Cnt  .set    00051h      ;Track crossing counter read port
ADC_Data    .set    00053h      ;MP87099 ADC Data
ADC_Addr    .set    00053h      ;MP87099 ADC Address
Clock_High  .set    00056h      ;Read to take ADC Clock High
Clock_Low   .set    00057h      ;Read to take ADC Clock Low
_Fine_DAC   .set    00058H      ;7228 fine current DAC
_Crs_DAC    .set    00059H      ;7228 Coarse current DAC
_ReadLS_DAC .set    0005AH      ;7228 Laser Read current DAC
_ReadMS_DAC .set    0005BH      ;7228 Laser Read current DAC
_Write_DAC  .set    0005CH      ;7228 Laser Write current DAC
_Foc_LS_DAC .set    0005DH      ;7228 focus current DAC
_Foc_MS_DAC .set    0005EH      ;7228 focus current DAC
_Spare_DAC  .set    0005FH      ;7228 Spin current DAC


Bit0        .set    15          ;Bit zero in BIT test is a 15
Bit1        .set    14          ;Bit 1 in BIT test is a 14
Bit2        .set    13
Bit3        .set    12
```

```
Bit4        .set    11
Bit5        .set    10
Bit6        .set    9
Bit7        .set    8
Bit8        .set    7
Bit9        .set    6
Bit10       .set    5
Bit11       .set    4
Bit12       .set    3
Bit13       .set    2
Bit14       .set    1
Bit15       .set    0           ;Bit 15 in BIT test is a 0

SBL         .set    5           ;Status Buffer Length
CBL         .set    10          ;Command Buffer Length

; Analog to Digital Converter constants
Focus_ADC   .set    00000H      ;Address zero
QSum_ADC    .set    00001H      ;Address one
Crs_ADC     .set    00002H      ;Address two
Fine_ADC    .set    00003H      ;Address three
FwdSen_ADC  .set    00004H      ;Address four
Test_ADC    .set    00005H      ;Address five

MaxPos      .set    07FFFH      ;Max Positive value for DAC images
MaxNeg      .set    08000H      ;Max Negative value for DAC images
Min_QSum    .set    00240H      ;Minimum QSum before switching refs
Max_Bad_Samples .set 00004H     ;Number of bad samples before Errors Flags

Ref_Select  .set    00004H      ;The reference select bit of Ctrl_Port


;     Timer interupt variables
_Count_20_LSW .usect Time_Ram,1   ;20us counter LSWord, incremented by timer
_Count_20_MSW .usect Time_Ram,1   ;20us counter MSWord, incremented by timer
_Int_QSum     .usect Time_Ram,1   ;Interrupt Quad Sum Value
_Int_Focus    .usect Time_Ram,1   ;Interrupt Focus Value
```

```
Focus_Error     .usect Time_Ram.1   :_Int_Focus - Zero Offset
Old_Focus_1     .usect Time_Ram.1   :and old focus value
Old_Focus_2     .usect Time_Ram.1   :and old focus value
_FocLS_Image    .usect Time_Ram.1   :Focus LS DAC memory image for capture
_FocMS_Image    .usect Time_Ram.1   :Focus DAC memory image for servos
Old_FocDac      .usect Time_Ram.1   :Focus DAC memory image
Focus_N1        .usect Time_Ram.1   :Focus Loop Constants. Numerator z^0
Focus_N2        .usect Time_Ram.1   :Focus Loop Constants. Numerator z^-1
Focus_N3        .usect Time_Ram.1   :Focus Loop Constants. Numerator z^-2
Focus_D2        .usect Time_Ram.1   :Focus Loop Constants. Denominator z^-1
Focus_D3        .usect Time_Ram.1   :Focus Loop Constants. Denominator z^-2
Focus_G         .usect Time_Ram.1   :Focus Loop Constants. Gain
_Foc_Err_Cnt    .usect Time_Ram.1   :Focus Sample out of spec counter
_Focus_Limit    .usect Time_Ram.1   :Focus Error In Spec Limit
_Int_Fine       .usect Time_Ram.1   :Interrupt Fine Value
Fine_Error      .usect Time_Ram.1   :_Int_Fine - Zero Offset
Old_Fine_1      .usect Time_Ram.1   :and old Fine value
Old_Fine_2      .usect Time_Ram.1   :and old Fine value
_FineDacImage   .usect Time_Ram.1   :Fine DAC memory image
Old_FineDac     .usect Time_Ram.1   :Fine DAC memory image
_FineDacZero    .usect Time_Ram.1   :Fine DAC Zero or Seek Accel Value
Fine_N1         .usect Time_Ram.1   :Fine Loop Constants. Numerator z^0
Fine_N2         .usect Time_Ram.1   :Fine Loop Constants. Numerator z^-1
Fine_N3         .usect Time_Ram.1   :Fine Loop Constants. Numerator z^-2
Fine_D2         .usect Time_Ram.1   :Fine Loop Constants. Denominator z^-1
Fine_D3         .usect Time_Ram.1   :Fine Loop Constants. Denominator z^-2
Fine_G          .usect Time_Ram.1   :Fine Loop Constants. Gain
_Fine_Err_Cnt   .usect Time_Ram.1   :Fine Sample out of spec counter
_Fine_Limit     .usect Time_Ram.1   :Fine Error In Spec Limit
_Int_Crs        .usect Time_Ram.1   :Interrupt Coarse Value
Crs_Error       .usect Time_Ram.1   :_Int_Crs - Zero Offset
Old_Crs_1       .usect Time_Ram.1   :and old Coarse value
Old_Crs_2       .usect Time_Ram.1   :and old Coarse value
_CrsDacImage    .usect Time_Ram.1   :Coarse DAC memory image
Old_CrsDac      .usect Time_Ram.1   :Coarse DAC memory image
_CrsDacZero     .usect Time_Ram.1   :Coarse DAC Zero or Seek Accel Value
Crs_N1          .usect Time_Ram.1   :Coarse Loop Constants. Numerator z^0
```

```
Crs_N2          .usect Time_Ram,1    :Coarse Loop Constants, Numerator z^-1
Crs_N3          .usect Time_Ram,1    :Coarse Loop Constants, Numerator z^-2
Crs_D2          .usect Time_Ram,1    :Coarse Loop Constants, Denominator z^-1
Crs_D3          .usect Time_Ram,1    :Coarse Loop Constants, Denominator z^-2
Crs_G           .usect Time_Ram,1    :Coarse Loop Constants, Gain
Pin_G           .usect Time_Ram,1    :Pinning Loop Constant, Gain
_Int_FwdSen     .usect Time_Ram,1    :Interrupt Forward Sense Value
_Int_Test       .usect Time_Ram,1    :Interrupt Test Value
_FwdSen_Zero    .usect Time_Ram,1    :ADC Forward Sense Value with the laser off
_QSum_Zero      .usect Time_Ram,1    :ADC Quad Sum Value with the laser off
_Fine_Zero      .usect Time_Ram,1    :ADC Tracking Value with the laser off
_Focus_Zero     .usect Time_Ram,1    :ADC Focus Value with the laser off
_Crs_Zero       .usect Time_Ram,1    :ADC Postion Error zero value
Sign_Bit        .usect Time_Ram,1    :Sign Bit value stored here (08000h)
Temp_1          .usect Time_Ram,1    :Temp Value
Temp_2          .usect Time_Ram,1    :Temp Value
_Debug_Ram      .usect Time_Ram,50   :Debug Ram Area


:     Command interupt variables
OldAR           .usect Params,1      :Old AR storage location for interupts
_Cmd_Bits       .usect Params,1      :Register to signal a command is ready
                                     :Bit 0 - 1 = Command Ready
                                     :Bit 1 - Old Direction Bit
                                     :Bit 2 - 0 = LSByte, 1 = MSByte
                                     :Bit 3 - Focus Sample available
                                     :Bit 4 - Fine Sample available
                                     :Bit 5 - Coarse Sample available
                                     :Bit 6 - Quad Sum Sample available
                                     :Bit 7 - Laser Sense Sample available
                                     :Bit 8 - Fine I/Test Sample available
                                     :Bit 9 - Tach Pulse happened

_Ctrl_Image     .usect Params,1      :memory image of the Control Port
_Cmd_Buff_Point .usect Params,1      :Storage for cmd pointers

_Stat_Buffer    .usect Time_Ram,SBL  :Status Buffer, Length = SBL
```

```
_CMD_Buffer     .usect Time_Ram.CBL   :Command Buffer. Length = CBL

Old_Tach_Time   .usect Params.1       :Last Tach pulse time tick
_Tach_Time      .usect Params.1       :Delta Tach Time
_TachUpLimit    .usect Params.1       :Tach pulse Upper time limit
_TachLowLimit   .usect Params.1       :Tach pulse Lower time limit

          .text
ISR1      RETE                        :INIT1- Track Crossing Signal
ISR2      RETE                        :INIT2- (Should not happen)
:*****************************************************************************
:  Spindle Motor Tach Interrupt Handler. Set the One_Rev bit
:*****************************************************************************
Tach                                   :INIT3- (Tach Pulse)
          LAMM   _Cmd_Bits            :Tell Kernel a Tach Pulse Happened
          OR     #00200h              :Set the Tach Pulse bit
          SAMM   _Cmd_Bits            :Save the new command bits
          LDPK   _Count_20_LSW        :Point to the timer interupt page
          LACL   _Count_20_LSW        :Get the current timer value
          LDPK   Old_Tach_Time        :Point to the Tach time
          SUB    Old_Tach_Time        :Subtract the old Value
          SACL   _Tach_Time           :Save the delta value
          ADD    Old_Tach_Time        :Restore the new time value
          SACL   Old_Tach_Time        :Save it in the old value
          RETE
:*****************************************************************************
:  Timer Interrupt Handler. All Real Time Servos and ADC's are done here!
:*****************************************************************************
MSW_Time_Tick                          :Only do this if lscounter rolled over
          LACL   _Count_20_MSW        :Increment the 20 us counter          1
          ADD    #1                   :                                     1
          SACL   _Count_20_MSW        :Save it                              1
          B      Test_Time_Odd        :Conitnue                             4
:  Actual start of the timer interupt routine
Timer
:debug
          LAMM   _Ctrl_Image          :Start of interrupt
```

```
                OR      #02
                SAMM    _Ctrl_Port
                SAMM    _Ctrl_Image             :Start of interrupt
        :debug
        :
        :  Start of  the Focus ADC Conversion and Compensation Loop
        :
        Focus_Start
        : Do the Focus Loop
                LACL    #Focus_ADC              :Load the address
                SAMM    ADC_Addr                :Tell the converter conversion address
        : Update the counter while waiting for the conversion
                LDPK    _Count_20_LSW           :Point to the timer interupt page     2
                LACC    _Count_20_LSW           :Increment the 20 us counter          1
                ADD     #1                      :                                     1
                SACL    _Count_20_LSW           :                                     1
                BCND    MSW_Time_Tick,EQ        :If Counter rolled over inc MSWord    2
        Test_Time_Odd
                BIT     _Count_20_LSW,Bit0      :See if even or odd count             1
                BCND    ODD_Count,TC            :Odd-Fwd Sense. Even-Coarse Loop    2,4
                LAMM    _Cmd_Bits               :Update Command Bits with Sample Data 1
                OR      #00038h                 :Coarse. Fine. Focus available        2
                SAMM    Clock_High              :Take the converter clock high        1
                B       Foc_B_1                 :Do the Even Stuff                    4
        ODD_Count
                NOP                             :One extra clock for ADC timing       1
                SAMM    Clock_High              :Take the converter clock high        1
                LAMM    _Cmd_Bits               :Update Command Bits with Sample Data 1
                OR      #000D8h                 :Sense. Fine. Focus. QSum available   2
                NOP                             :One extra clock for ADC timing       1
        Foc_B_1
                SAMM    Clock_Low               :Take the converter clock low         1
                SAMM    _Cmd_Bits               :Save the results                     1
                NOP
                SAMM    Clock_High              :Take the converter clock high        2
                LAMM    ADC_Data                :Get the result of the conversion     1
                XOR     Sign_Bit                :Make it a signed integer             1
```

```
                AND   #0FFC0h           :Clear the 6 LSBits
                SACL  _Int_Focus        :Save Focus Error                1
        : Debug
                XOR   Sign_Bit          :Add in the zero offset (8000H)  1
                SAMM  _Spare_DAC
                XOR   Sign_Bit          :Add in the zero offset (8000H)  1
        : Debug
                SUB   _Focus_Zero       :Save the Focus Error            1
                SACL  Focus_Error       :Focus DAC - Zero Value          1
                SPM   1                 :Set for fraction multiply       1
                ZAP                     :                                1
                LT    Old_FocDac        :Vout(N-2)                       1
                MPY   Focus_D3          :Multiply by a constant          1
                LTD   _FocMS_Image      :Vout(N-1) > Vout(N-2)           1
                MPY   Focus_D2          :Multiply by a constant          1
                APAC                    :Accumulate the results          1
                SACH  Temp_1            :(D2*Vout(N-1)+D3*Vout(N-2) > temp 1
                ZAP                     :                                1
                LT    Old_Focus_2       :Vin(N-2)                        1
                MPY   Focus_N3          :Multiply by a constant          1
                LTD   Old_Focus_1       :Vi(N-1)>Vi(N-2) acc=Vi(n-2)*N3  1
                MPY   Focus_N2          :Multiply by a constant          1
                LTD   Focus_Error       :Vi(N)>Vi(N-1) acc=acc+Vi(n-1)*N2 1
        : Focus Error values are updated. Update the focus DAC if the loop is closed
                BIT   _Stat_Buffer.Bit8 :See if the focus loop is closed 1
                BCND  Focus_Open.NTC    :Branch if the loop is not closed 2
        :
                MPY   Focus_N1          :preg = Vi(n)*N1                 1
                APAC                    :Acc = Acc + Vi(n)*N1            1
                SACH  Temp_2            :Save result                     1
                LT    Temp_2            :Load the TReg                   1
                SPM   0                 :Set for regular multiply        1
                MPY   Focus_G           :PReg = K2*(K3*Vin(N-1) - Vin(N)) 1
                LACC  Temp_1            :acc = (D2*Vout(N-1)+D3*Vout(N-2) 1
                APAC                    :acc = G*Numerator-Denominator   1
                XOR   Sign_Bit          :Add in the zero offset (8000H)  1
                SAMM  _Foc_MS_DAC       :Write out the value             1
```

```
                XOR     Sign_Bit            :Set the bit back                   1
                SACL    _FocMS_Image        :Save the image                     1
        :Loop done unless there is an overflow
                BSAR    15                  :sign extend the 32 bit number      1
 5              BCND    Foc_Pos_OV.GT       :If greater than zero. + overflow   2
                CMPL                        :Ones complement the Acc            1
                BCND    Foc_Neg_OV.GT       :If greater than zero. - overflow   2
                B       Fine_Start          :Do the quad sum loop               4
        Foc_Pos_OV
10              LACC    #MaxPos             :Write Out Max Positive value       2
                SACL    _FocMS_Image        :Save the image                     1
                XOR     Sign_Bit            :Add in the zero offset (8000H)     1
                SAMM    _Foc_MS_DAC         :Write out the value                1
                B       Fine_Start          :                                   4
15      Foc_Neg_OV
                LACC    #MaxNeg             :Write Out Max Positive value       2
                SACL    _FocMS_Image        :Save the image                     1
                XOR     Sign_Bit            :Add in the zero offset (8000H)     1
                SAMM    _Foc_MS_DAC         :Write out the value                1
20              B       Fine_Start          :                                   4
        :
        :  End of  the Focus ADC Conversion and Compensation Loop
        :  Start of the Fine Loop ADC Conversion and Compensation Loop
        :
25      Focus_Open                          :Focus Loop Open. No Error Check
                LACL    #Fine_ADC           :TES address                        1
                SAMM    ADC_Addr            :Tell the converter conversion address3
                RPT     #4                  :Delay 6 clocks                     6
                NOP
30              B       Fine_Clk_High       :4 more clock of delay              4
        Fine_Start
        : Do the Fine Tracking Loop
                LACL    #Fine_ADC           :TES address                        1
                SAMM    ADC_Addr            :Tell the converter conversion address3
35              LACC    Focus_Error         :Get the Focus Error Value          1
                ABS                         :Get the magnitude                  1
                SUB     _Focus_Limit        :Compare to out of focus limit      1
```

```
        BCND    Focus_In_Spec.LEQ    :Branch if in spec                  4/2
Focus_Out_Spec
        LACL    _Foc_Err_Cnt         :Get The Focus Error Count          1
        SUB     #Max_Bad_Samples     :See if at max Number of bad Samples 1
        BCND    Fine_Clk_High.GEQ    :If at Max then continue            4/2
        LACL    _Foc_Err_Cnt         :Get The Focus Error Count          1
        ADD     #1                   :Increment the Count                1
        SAMM    Clock_High           :Take the converter clock high      1
        SACL    _Foc_Err_Cnt         :Save the Incremeted Value          1
        SUB     #Max_Bad_Samples     :See if at max Number of bad Samples 1
        BCND    Fine_Clk_Low.LT      :Continue if not at max count       4/2
        SAMM    Clock_Low            :Take the converter clock low       1
        LACL    _Stat_Buffer         :Set the Bad Focus Bit              1
        OR      #0400H               :Set the Bit                        2
        SACL    _Stat_Buffer         :Save the Result                    1
: Debug. In the future. set the 188 interupt here
        B       Fine_Clk_H2          :Continue                           4
Dec_Foc_Cnt
        SAMM    Clock_High           :Take the converter clock high      1
        SUB     #1                   :Decrement the Focus Error Count    1
        SACL    _Foc_Err_Cnt         :Save the Decremeted Value          1
        NOP                          :Delay 1 Clock                      1
        NOP                          :Delay 1 Clock                      1
        SAMM    Clock_Low            :Take the converter clock low       1
        B       Fine_Clk_H2          :Take the Clock Bit Low             4
Focus_In_Spec
        LACL    _Foc_Err_Cnt         :Get the Focus Error Count          1
        BCND    Dec_Foc_Cnt.NEQ      :If not Zero then decrement         4/2
        NOP
Fine_Clk_High
        SAMM    Clock_High           :Take the converter clock high      1
        RPT     #1                   :Delay 3 clocks                     3
        NOP
Fine_Clk_Low
        SAMM    Clock_Low            :Take the converter clock low       1
        NOP                          :                                   1
        NOP                          :                                   1
```

```
Fine_Clk_H2
        SAMM    Clock_High          :Take the converter clock high      2
        LAMM    ADC_Data            :Get the Data                       3
        XOR     Sign_Bit            :Make it a signed integer           1
        SACL    _Int_Fine           :Save Fine Error Value              1
        SUB     _Fine_Zero          :Save the Fine Error                1
        SACL    Fine_Error          :Fine Error - Zero Value            1
:Set up for fixed Reference Conversions
        LAMM    _Ctrl_Image         :                                   1
        OR      #Ref_Select         :Fixed Reference Bit                2
        SAMM    _Ctrl_Image         :                                   1
        SAMM    _Ctrl_Port          :                                   1
        SPM     1                   :Set for fraction multiply          1
:Do the Loop Compensation
        ZAP                         :                                   1
        LT      Old_FineDac         :Vout(N-2)                          1
        MPY     Fine_D3             :Multiply by a constant             1
        LTD     _FineDacImage       :Vout(N-1) > Vout(N-2)              1
        MPY     Fine_D2             :Multiply by a constant             1
        APAC                        :Accumulate the results             1
        SACH    Temp_1              :(D2*Vout(N-1)+D3*Vout(N-2) > temp  1
        ZAP                         :                                   1
        LT      Old_Fine_2          :Vin(N-2)                           1
        MPY     Fine_N3             :Multiply by a constant             1
        LTD     Old_Fine_1          :Vi(N-1)>Vi(N-2) acc=Vi(n-2)*N3     1
        MPY     Fine_N2             :Multiply by a constant             1
        LTD     Fine_Error          :Vi(N)>Vi(N-1) acc=acc+Vi(n-1)*N2   1
:Fine Error values are updated. Update the fine DAC if the loop is closed
        BIT     _Stat_Buffer.Bit7   :See if the fine loop is closed     1
        BCND    FwdSenOrCrs.NTC     :Branch if the loop is not closed   2
        MPY     Fine_N1             :preg = Vi(n)*N1                    1
        APAC                        :Acc = Acc + Vi(n)*N1               1
        SACH    Temp_2              :Save result                        1
        LT      Temp_2              :Load the TReg                      1
        SPM     0                   :Set for regular multiply           1
        MPY     Fine_G              :PReg = K2*(K3*Vin(N-1) - Vin(N))   1
        LACC    Temp_1              :acc = (D2*Vout(N-1)+D3*Vout(N-2)   1
```

```
            APAC                    :acc = G*Numerator-Denominator      1
            ADD     _FineDacZero    :Add the DAC Zero Value(or Seek Accel)1
            SACL    _FineDacImage   :Save the image                     1
            XOR     Sign_Bit        :Toggle the MSBit                   1
            SAMM    _Fine_DAC       :Write out the value                1
            XOR     Sign_Bit        :Toggle the MSBit back              1
             1
    :Loop done unless there is an overflow
            BSAR    15              :sign extend the 32 bit number      1
            BCND    Fine_Pos_OV.GT  :If greater than zero. + overflow   2
            CMPL                    :Ones complement the Acc            1
            BCND    Fine_Neg_OV.GT  :If greater than zero. - overflow   2
            B       FwdSenOrCrs     :Do the quad sum loop               4
    Fine_Pos_OV
            LACC    #MaxPos         :Write Out Max Positive value       2
            SACL    _FineDacImage   :Save the image                     1
            XOR     Sign_Bit        :Toggle the MSBit                   1
            SAMM    _Fine_DAC       :Write out the value                1
            B       FwdSenOrCrs     :                                   4
    Fine_Neg_OV
            LACC    #MaxNeg         :Write Out Max Positive value       2
            SACL    _FineDacImage   :Save the image                     1
            XOR     Sign_Bit        :Toggle the MSBit                   1
            SAMM    _Fine_DAC       :Write out the value                1
    :
    :  End of  the Fine ADC Conversion and Compensation Loop
    :
    Fwd_Sen_Addr
    : Get the Quad Sum Value
            LACC    #QSum_ADC       :Get the Quad Sum Address
            SAMM    ADC_Addr        :Tell the converter to start
            B       FwdSen_Crs_ADC  :Continue Conversion/Error Checking
    :
    FineClosed      :Error Check starts here if the Fine loop is closed
            LACC    Fine_Error      :Get the Fine Error Value           1
            ABS                     :Get Magnitude of the error
            SAMM    Clock_High      :Take the converter clock high      1
```

```
                SUB     _Fine_Limit         ;Subtract the Fine error limit
                BCND    Fine_Out_Spec.GT    ;Branch if out of Spec
        Fine_In_Spec
                LACL    _Fine_Err_Cnt       ;Get the error count                1
                SAMM    Clock_Low           ;Take the converter clock low       1
                BCND    Dec_Fine_Cnt.NEQ    ;If not Equal then Decrement        2
                SAMM    Clock_High          ;Take the converter clock high      2
                LAMM    ADC_Data            ;Get the Data                       3
                SACL    Temp_2              ;Save the Results in temp 2
                B       Finish_Fwd_Crs      ;Finish the forward sense or coarse
                :
        Dec_Fine_Cnt
                SUB     #1                  ;If not Equal then Decrement        1
                SACL    _Fine_Err_Cnt       ;Save the Result                    1
                SAMM    Clock_High          ;Take the converter clock high      2
                LAMM    ADC_Data            ;Get the Data                       3
                SACL    Temp_2              ;Save the Results in temp 2
                B       Finish_Fwd_Crs      ;Finish the forward sense or coarse 4
                :
        Fine_Out_Spec
                SAMM    Clock_Low           ;Take the converter clock low       1
                NOP
                NOP
                SAMM    Clock_High          ;Take the converter clock high      2
                LAMM    ADC_Data            ;Get the Data                       3
                SACL    Temp_2              ;Save the Results in temp 2         1
                LACL    _Fine_Err_Cnt       ;Get the error count                1
                SUB     #Max_Bad_Samples    ;See if at max number of bad samples 1
                BCND    Finish_Fwd_Crs.GEQ  ;If at max then continue          4/2
                LACL    _Fine_Err_Cnt       ;else increment the error count     1
                ADD     #1                  ;increment                          1
                SACL    _Fine_Err_Cnt       ;Save the count                     1
                SUB     #Max_Bad_Samples    ;Test if now at max count           1
                BCND    Finish_Fwd_Crs.LT   ;continue if not a max count      4/2
                LACL    _Stat_Buffer        ;else set the fine tracking error bit 1
                OR      #1000H              ;Set the Bit                        2
                SACL    _Stat_Buffer        ;Save the result                    1
```

```
; Debug. in the Future set the 188 interupt bit here
        B       Finish_Fwd_Crs      ;Finish the forward sense or coarse
;
FwdSenOrCrs
; Do the Coarse Or Forward Sense Conversion
        BIT     _Count_20_LSW.Bit0  ;See if even or odd count
        BCND    Fwd_Sen_Addr.TC     ;Odd-Fwd Sense. Even-Coarse Loop
; If not forward sense loop then do the coarse loop
        LACC    #Crs_ADC            ;Get the Coarse conversion command
        SAMM    ADC_Addr            ;Tell the converter to start
        RPT     #1
        NOP
FwdSen_Crs_ADC                      ;Start fine loop error checking !!!
        BIT     _Stat_Buffer.Bit7   ;See if the fine loop is closed      1
        BCND    FineClosed.TC       ;Branch if the loop is closed      4/2
        RPT     #1                  ;No error checking if loop is open
        NOP
        SAMM    Clock_High          ;Take the converter clock high       1
        RPT     #1
        NOP
        SAMM    Clock_Low           ;Take the converter clock low        1
        NOP
        NOP
        SAMM    Clock_High          ;Take the converter clock high       2
        LAMM    ADC_Data            ;Get the Data                        3
        SACL    Temp_2              ;Save the value
Finish_Fwd_Crs
        LACC    Temp_2              ;Restore data
        BIT     _Count_20_LSW.Bit0  ;See if even or odd count
        BCND    Fwd_Sense.TC        ;Odd-Fwd Sense. Even-Coarse Loop
        XOR     Sign_Bit            ;Make it a signed integer            1
        SACL    _Int_Crs            ;Save Coarse Error Value             1
        SUB     _Crs_Zero           ;Subtract the zero value             1
        SACL    Crs_Error           ;Crs DAC - Zero Value                1
        BIT     _Stat_Buffer.Bit5   ;See if the Pinning loop is close    1
        BCND    DoPin.TC            ;Branch if pinning is closed         2
        SPM     1                   ;Set for fraction multiply           1
```

```
                ZAP                             :                                       1
                LT      Old_CrsDac              :Vout(N-2)                              1
                MPY     Crs_D3                  :Multiply by a constant                 1
                LTD     _CrsDacImage            :Vout(N-1) > Vout(N-2)                  1
                MPY     Crs_D2                  :Multiply by a constant                 1
                APAC                            :Accumulate the results                 1
                SACH    Temp_1                  :(D2*Vout(N-1)+D3*Vout(N-2) > temp      1
                ZAP                             :                                       1
                LT      Old_Crs_2               :Vin(N-2)                               1
                MPY     Crs_N3                  :Multiply by a constant                 1
                LTD     Old_Crs_1               :Vi(N-1)>Vi(N-2) acc=Vi(n-2)*N3         1
                MPY     Crs_N2                  :Multiply by a constant                 1
                LTD     Crs_Error               :Vi(N)>Vi(N-1) acc=acc+Vi(n-1)*N2       1
        : Crs Error values are updated. Update the Crs DAC if the loop is closed
                BIT     _Stat_Buffer.Bit6       :See if the Coarse loop is closed       1
                BCND    SwitchRef.NTC           :Branch if the loop is not closed       2
        :
                MPY     Crs_N1                  :preg = Vi(n)*N1                        1
                APAC                            :Acc = Acc + Vi(n)*N1                   1
                SACH    Temp_2                  :Save result                            1
                LT      Temp_2                  :Load the TReg                          1
                SPM     0                       :Set for regular multiply               1
                MPY     Crs_G                   :PReg = K2*(K3*Vin(N-1) - Vin(N))       1
                LACC    Temp_1                  :acc = (D2*Vout(N-1)+D3*Vout(N-2)       1
                APAC                            :acc = (G*Numerator-Denominator)/4      1
                SFL                             :Shift left. Multiply by 2              1
                SFL                             :Shift left. Multiply by 4              1
                ADD     _CrsDacZero             :Add the DAC Zero Value                 1
                XOR     Sign_Bit                :Add in the zero offset (8000H)         1
                SAMM    _Crs_DAC                :Write out the value                    1
                XOR     Sign_Bit                :Set the bit back                       1
                SACL    _CrsDacImage            :Save the image                         1
        :Loop done unless there is an overflow
                BSAR    15                      :sign extend the 32 bit number          1
                BCND    Crs_Pos_OV.GT           :If greater than zero, + overflow       2
                CMPL                            :Ones complement the Acc                1
                BCND    Crs_Neg_OV.GT           :If greater than zero, - overflow       2
```

```
        B       SwitchRef       :Do the quad sum loop                   4
Crs_Pos_OV
        LACC    #MaxPos         :Write Out Max Positive value           2
        SACL    _CrsDacImage    :Save the image                         1
        XOR     Sign_Bit        :Add in the zero offset (8000H)         1
        SAMM    _Crs_DAC        :Write out the value                    1
        B       SwitchRef       :                                       4
Crs_Neg_OV
        LACC    #MaxNeg         :Write Out Max Positive value           2
        SACL    _CrsDacImage    :Save the image                         1
        XOR     Sign_Bit        :Add in the zero offset (8000H)         1
        SAMM    _Crs_DAC        :Write out the value                    1
        B       SwitchRef       :                                       4
; Crs Error values are updated. Do the Pinning Loop
DoPin
        NEG                     :Negate the Coarse Error
        SACL    Crs_Error       :Crs DAC - Zero Value                   1
        SPM     1               :Set for fraction multiply              1
        ZAP                     :                                       1
        LT      Old_FineDac     :Vout(N-2)                              1
        MPY     Crs_D3          :Multiply by a constant                 1
        LTD     _FineDacImage   :Vout(N-1) > Vout(N-2)                  1
        MPY     Crs_D2          :Multiply by a constant                 1
        APAC                    :Accumulate the results                 1
        SACH    Temp_1          :(D2*Vout(N-1)+D3*Vout(N-2) > temp      1
        ZAP                     :                                       1
        LT      Old_Crs_2       :Vin(N-2)                               1
        MPY     Crs_N3          :Multiply by a constant                 1
        LTD     Old_Crs_1       :Vi(N-1)>Vi(N-2) acc=Vi(n-2)*N3         1
        MPY     Crs_N2          :Multiply by a constant                 1
        LTD     Crs_Error       :Vi(N)>Vi(N-1) acc=acc+Vi(n-1)*N2       1
;
        MPY     Crs_N1          :preg = Vi(n)*N1                        1
        APAC                    :Acc = Acc + Vi(n)*N1                   1
        SACH    Temp_2          :Save result                            1
        LT      Temp_2          :Load the TReg                          1
        SPM     0               :Set for regular multiply               1
```

```
        MPY     Pin_G               :PReg = K2*(K3*Vin(N-1) - Vin(N))     1
        LACC    Temp_1              :acc = (D2*Vout(N-1)+D3*Vout(N-2)     1
        APAC                        :acc = (G*Numerator-Denominator)/4    1
        SFL                         :Shift left, Multiply by 2            1
        SFL                         :Shift left, Multiply by 4            1
        ADD     _FineDacZero        :Add the DAC Zero Value               1
        XOR     Sign_Bit            :Add in the zero offset (8000H)       1
        SAMM    _Fine_DAC           :Write out the value                  1
        XOR     Sign_Bit            :Set the bit back                     1
        SACL    _FineDacImage       :Save the image                       1
;Loop done unless there is an overflow
        BSAR    15                  :sign extend the 32 bit number        1
        BCND    Pin_Pos_OV.GT       :If greater than zero, + overflow     2
        CMPL                        :Ones complement the Acc              1
        BCND    Pin_Neg_OV.GT       :If greater than zero, - overflow     2
        B       SwitchRef           :Do the quad sum loop                 4
Pin_Pos_OV
        LACC    #MaxPos             :Write Out Max Positive value         2
        SACL    _FineDacImage       :Save the image                       1
        XOR     Sign_Bit            :Add in the zero offset (8000H)       1
        SAMM    _Fine_DAC           :Write out the value                  1
        B       SwitchRef           :                                     4
Pin_Neg_OV
        LACC    #MaxNeg             :Write Out Max Positive value         2
        SACL    _FineDacImage       :Save the image                       1
        XOR     Sign_Bit            :Add in the zero offset (8000H)       1
        SAMM    _Fine_DAC           :Write out the value                  1
        B       SwitchRef           :                                     4
; Read the forward sense and quad sum values
Fwd_Sense
        SACL    _Int_QSum           :Save the Quad Sum Value
; Get the forward sense value
        LACC    #FwdSen_ADC         :Get the Forward Sense Address
        SAMM    ADC_Addr            :Tell the converter to start
        LACL    _Int_QSum           :Load it for modification
        BSAR    6                   :Make it a 10 bit value
        AND     #03FFH              :Make it positive
```

```
                SACL    _Int_QSum           ;Save the Forward Sense Value
                RPT     #3
                NOP
                SAMM    Clock_High          ;Take the converter clock high      1
                RPT     #1
                NOP
                SAMM    Clock_Low           ;Take the converter clock low       1
                NOP
                NOP
                SAMM    Clock_High          ;Take the converter clock high      2
                LAMM    ADC_Data            ;Get the result of the conversion
                BSAR    6                   ;Make it a 10 bit value
                AND     #03FFH              ;Make it positive
                SACL    _Int_FwdSen         ;Save the Forward Sense Value
    SwitchRef
                LAMM    _Ctrl_Image         ;Get the control image
                AND     #0008h              ;See if the laser is on
                BCND    TimeEnd.EQ          ;If laser is off don't switch ref
                LAMM    _Ctrl_Image         ;Set up for Quad Sum Reference
                AND     #-Ref_Select        ;Clear the bit
                SAMM    _Ctrl_Port          ;Write it out
                SAMM    _Ctrl_Image
    TimeEnd
    :debug
                LAMM    _Ctrl_Image         ;end of interrupt
                AND     #0FDh
                SAMM    _Ctrl_Port
                SAMM    _Ctrl_Image
    :debug
                RETE                        ;Timer interupt processing
    :
    RCV         RETE                        ;Serial Rx (Should not happen)
    XMT         RETE                        ;Serial Xmit (Should not happen)
    TDMRCV      RETE                        ;TDM Serial Rx (Should not happen)
    TDMXMT      RETE                        ;TDM Serial Xmit (Should not happen)
    ;*************************************************************************
    :  Command Interupt Handler, Interrupt 4
```

```
;*************************************************************************
CMD_Intr
          BCND   CMD_Stat.BIO          ;If BIO=0.then see is a command or stat
          SETC   XF                    ;else Set the XF Bit and
          RETE                         ;Return
CMD_Stat
          LDP    #000h                 ;Point to page 0;
          BIT    TSPC.Bit8             ;Test Bit 8 of the TSPC (Direction)
          BCND   Dir_Eq_High.TC        ;If Direction is 1 the See if Old = 1
          BIT    _Cmd_Bits.Bit1        ;Test the Old Direction bit
          BCND   CMD_Complete.TC       ;Branch if Old Dir = 1 & Dir = 0
;
Next_Status                            ;Old Dir=0.Dir=0 => ship next status;
          BIT    _Cmd_Bits.Bit2        ;Test the send MSByte or LSByte
          BCND   Stat_MSB.TC           ;Branch MSByte then branch
Stat_LSB
;Note Old Status bit is set
          MAR    *.AR1                 ;Make AR1 active
          SAR    AR1.OldAR             ;Save AR1
          LAR    AR1._Cmd_Buff_Point   ;Load the pointer into AR1
          LACC   *+                    ;Load Status into the Accumulator
          SAMM   Cmd_Port              ;Write Out the Status;
          SAR    AR1._Cmd_Buff_Point   ;Save The Status Pointer
          LAR    AR1.#_Cmd_Bits        ;Toggle the MSByte/LSByte Bit
          XPL    #00004h.*             ;Toggle Bit 2 of the Cmd_Bits
          LAR    AR1.OldAR             ;Restore AR1
          CLRC   XF                    ;Clear the Acknowledge Bit
          RETE                         ;Exit interupt 4
Stat_MSB
          MAR    *.AR1                 ;Make AR1 active
          SAR    AR1.OldAR             ;Save AR1
          LAR    AR1._Cmd_Buff_Point   ;Load the pointer into AR1
          LACC   *.8                   ;Load Status shifted 8 into the Acc.
          SACH   Cmd_Port              ;Write Out the Status;
                                       ;The Status Pointer is unchanged
          LAR    AR1.#_Cmd_Bits        ;Toggle the MSByte/LSByte Bit
          XPL    #00004h.*             ;Toggle Bit 2 of the Cmd_Bits
```

```
                LAR     AR1,OldAR           :Restore AR1
                CLRC    XF                  :Clear the Acknowledge Bit
                RETE                        :Exit interupt 4
        :
Dir_Eq_High
                BIT     _Cmd_Bits,Bit1      :Test the Old Direction bit
                BCND    New_CMD,NTC         :Branch if Old Dir = 0 & Dir = 1
        Next_CMD                            :Old Dir=1,Dir=1 => get next cmd word:
                                            :Note Old Status bit is set
                BIT     _Cmd_Bits,Bit2      :Test the send MSByte or LSByte
                BCND    CMD_MSB,TC          :Branch MSByte then branch
        CMD_LSB
        :Note Old Status bit is set
                MAR     *,AR1               :Make AR1 active
                SAR     AR1,OldAR           :Save AR1
                LAR     AR1,_Cmd_Buff_Point :Load the pointer into AR1
                LAMM    Cmd_Port            :Load Command into the Accumulator
                AND     #00FFh              :Clear the MSBits
                OR      *                   :Or with the MSByte
                SACL    *                   :Save the command word
        :if AR1 > Command Buffer + Command Buffer Lenght then dec AR1
                LAR     AR1,#_Cmd_Bits      :Toggle the MSByte/LSByte Bit
                XPL     #00004h,*           :Toggle Bit 2 of the Cmd_Bits
                LAR     AR1,OldAR           :Restore AR1
                CLRC    XF                  :Clear the Acknowledge Bit
                RETE                        :Exit interupt 4
        New_CMD
                LAMM    _Cmd_Bits           :Get the Command Bit Register
                OR      #00006h             :Set the Old Dir and MSB/LSB bits
                SAMM    _Cmd_Bits           :Save the new command bits
                MAR     *,AR1               :Make AR1 active
                SAR     AR1,OldAR           :Save AR1
                LAR     AR1,#_CMD_Buffer    :Load AR1 to the start of Cmd Buffer
                MAR     *-,AR1              :Decrement the pointer
                SAR     AR1,_Cmd_Buff_Point :Save it in the pointer
                B       Save_CMD_MSB
        CMD_MSB
```

```
                MAR     *,AR1                   ;Make AR1 active
                SAR     AR1,OldAR               ;Save AR1
                LAR     AR1,_Cmd_Buff_Point     ;Load the pointer into AR1
        Save_CMD_MSB
        LACC    _Cmd_Buff_Point         ;Get the Pointer Value
        ;Subtract the start of the buffer + Length - 1
                SUB     #(_CMD_Buffer + CBL - 1)
                BCND    Save_Cmd_Point,GEQ      ;Branch if at the end of the buffer
                MAR     *+,AR1                  ;else increment the pointer
        Save_Cmd_Point
                SAR     AR1,_Cmd_Buff_Point     ;Save The Command Pointer
                LACC    Cmd_Port,8              ;Get the MSByte data in Acc. shl 8
                SACL    *                       ;Store the command into the buffer
                                                ;The Status Pointer is unchanged
                LAR     AR1,#_Cmd_Bits          ;Toggle the MSByte/LSByte Bit
                XPL     #00004h,*               ;Toggle Bit 2 of the Cmd_Bits
                LAR     AR1,OldAR               ;Restore AR1
                CLRC    XF                      ;Clear the Acknowledge Bit
                RETE                            ;Exit interupt 4
        ;
        CMD_Complete
                LAMM    _Cmd_Bits               ;Tell Kernel a command is ready
                OR      #00001h                 ;Set the Command Ready bit
                AND     #0FFFDh                 ;Clear Old Direction bit
                SAMM    _Cmd_Bits               ;Save the new command bits
                LAMM    IMR                     ;Get the Interupt Mask Register
                AND     #0FEFFh                 ;Clear interupt 4 enable bit
                SAMM    IMR                     ;Disable interupt 4
                RETE                            ;Exit interupt 4
        ;
        TRP     RETE                            ;Software trap (Should not happen)
        NMISR   RETE                            ;NMI interupt (Should not happen)
        ;*****************************************************************************
        ;   Program Name        : C50_Init.asm
        ;   Description         : DSP Initialization for 4x 5.25"
        ;   Part Number         : 562096
        ;   Date                : 8/12/93
```

```
;    O/S                : N/A
;    Compiler           : TI TMS320C2x/C5x Compiler, #TMDS3242855-02.Rel. 6.0
;    Support Packages   : N/A
;    Author             : Dave Schell
;    Required Files     : Drive.c,Interupt.asm,C50_init.asm,Seek.c,Drive.h
;                       : Recal.c
;    Hardware Required  : Part # XXXXXX
;    Install. Instr.    : Link in with Drive code
;    Operating Instr.   : N/A
;
;    Rev History
;      Date      Rev  C#  Init     Change Description
;    4/14/94     XA   00   DLS    Initial Release
;*******************************************************************************
 .include SimSet.equ

          .title  "Processor Initialization, C50_INIT.asm"
          .mmregs
          .ref    ISR1,ISR2,Tach,CMD_Intr,Timer
          .ref    RCV,XMT,TDMRCV,TDMXMT,TRP,NMISR,Sign_Bit
          .ref    _main,_c_int0,_Count_20_LSW,_Cmd_Bits,_Ctrl_Image
          .ref    _Stat_Buffer,_Count_20_MSW,_TachUpLimit,_TachLowLimit
          .ref    Focus_N1,Focus_N2,Focus_N3,Focus_D2,Focus_D3,Focus_G
          .ref    Fine_N1,Fine_N2,Fine_N3,Fine_D2,Fine_D3,Fine_G
          .ref    Crs_N1,Crs_N2,Crs_N3,Crs_D2,Crs_D3,Crs_G,Pin_G
          .ref    _FineDacZero,_CrsDacZero,_Foc_Err_Cnt,_Focus_Limit
          .ref    _Fine_Err_Cnt,_Fine_Limit
          .def    _init_regs,_Vel_Table,_InverseTime,_Read_Sense
          .def    _Write_Sense,_ReadMSImage,_ReadLSImage,_WriteDacImage
          .def    _MaxRPP,_MinRPP

_Vel_Table .usect V_Table,180h       ;Seek Velocity Table RAM Area
          .bss  _InverseTime,25      ;Inverse time table for seeks
          .bss  _Read_Sense,1        ;Laser Read Sense Desired Level
          .bss  _Write_Sense,1       ;Laser Write Sense Desired Level
          .bss  _ReadMSImage,1       ;Laser Read DAC Bit Image
          .bss  _ReadLSImage,1       ;Laser Read DAC Bit Image
```

```
                .bss  _WriteDacImage.1      ;Laser Write DAC 16 Bit Image
                .bss  _MaxRPP.1             ;Max RPP Value seen during a jumpback
                .bss  _MinRPP.1             ;Min RPP Value seen during a jumpback

5   ; Abs(N1)+Abs(N2)+Abs(N3) must be < 1 to prevent over flow
    ; Abs(D2)+Abs(D3) must be < 1 to prevent over flow

    :Vo(n)=-(D2*Vo(n-1)+D3*Vo(N-2))+(-1)*2*G*(0.5*(N1*Vi(N)+N2*Vi(N-1)+N3*Vi(N-3))
    Foc_N1      .set    -8192           :Foc Loop Const.-N1/4. -1.000*2^13
10  Foc_N2      .set    -877            :Foc Loop Const.-N2/4. -0.107*2^13
    Foc_N3      .set    7315            :Foc Loop Const.-N3/4. .893*2^13
    Foc_D2      .set    -11665          :Foc Loop Const.-D2. -.356*2^15
    Foc_D3      .set    -4456           :Foc Loop Const.-D3. -.136*2^15
    Foc_G       .set    167             :Foc Loop Gain Const. 4*36.059(*1.16??)
15
    :Vo(n)=-(D2*Vo(n-1)+D3*Vo(N-2))+(-1)*2*G*(0.5*(N1*Vi(N)+N2*Vi(N-1)+N3*Vi(N-3))
    Fin_N1      .set    8192            :Fin Loop Const. N1/4. 1.000*2^13
    Fin_N2      .set    877             :Fin Loop Const. N2/4. 0.107*2^13
    Fin_N3      .set    -7315           :Fin Loop Const. N3/4. -.893*2^13
20  Fin_D2      .set    -11665          :Fin Loop Const. -D2. -.356*2^15
    Fin_D3      .set    -4456           :Fin Loop Const. -D3. -.136*2^15
    Fin_G       .set    34              :Fin Loop Gain Const. 4*8.583

    :Vo(n)=-(D2*Vo(n-1)+D3*Vo(N-2))+(-1)*2*G*(0.5*(N1*Vi(N)+N2*Vi(N-1)+N3*Vi(N-3))
25  Cr_N1       .set    8192            :Crs Loop Const.-N1/2. (1.000/4)*2^15
    Cr_N2       .set    203             :Crs Loop Const.-N2/2. (0.02482/4)*2^15
    Cr_N3       .set    -7989           :Crs Loop Const.-N3/2. (-.97518/4)*2^15
    Cr_D2       .set    11893           :Crs Loop Const.-D2.   (1.4518/4)*2^15
    Cr_D3       .set    -4703           :Crs Loop Const.-D3.   (-.57412/4)*2^15
30  Cr_G        .set    7               :Crs Loop Gain Const. 6.991

    Pn_G        .set    7               :Pinning Loop Gain Const. 7.309

    V_TBL       .sect   "vectors"
35
    RESET       B       _c_int0             ;This section will be loaded in
                                            ;program memory address 0h.
```

```
INT1        B       ISR1                    :INIT1- begins processing here
INT2        B       ISR2                    :INIT2- begins processing here
INT3        B       Tach                    :INIT3- Spindle Motor Tach intr
TINT        B       Timer                   :Timer interupt processing
RINT        B       RCV                     :Serial Recieve processing
XINT        B       XMT                     :Serial transmit processing
TRNT        B       TDMRCV                  :TDM Serial Recieve processing
TXNT        B       TDMXMT                  :TDM Serial transmit processing
INT4        B       CMD_Intr                :INIT4- begins processing here
            .space  14*16                   :14 words
TRAP        B       TRP                     :Software trap processing
NMI         B       NMISR                   :NMI interupt processing

            .text
_init_regs  CLRC    OVM                     :Allow nornal Overflow in Acc
            LDP     #00h                    :Load the Data Pointer to 00h
 .if Simulator = 1
: Need the next line to work with the simulator only.
            SPLK    #0800h.PMST             :IPTR = 0800h. clear the rest
: Need this to work with real system.
 .else
            SPLK    #0810h.PMST             :Put SARAM into program memory
                                            :and set IPTR = 0800h
 .endif
            SPLK    #00h.CWSR               :Wait states are small and short
            SPLK    #00h.PDWSR              :Set 0 wait states for ext mem
            SPLK    #0FF09h.IOWSR           :0.1.& 2 wait states at 40MHz
            SPLK    #399.PRD                :20uS period timer. (50ns*(400-1))
            SPLK    #20h.TCR                :Reload and Enable the timer
            SPLK    #010Ch.IMR              :Disable all interupts except the
                                            :timer. tach (Intr 3) and
                                            :Command Interupt (Intr 4)
            ZAP                             :Zero the accumulator
            LDP     #_Count_20_LSW          :Point to the memory location
            SACL    _Count_20_LSW           :Zero the 20us count registar:
            LDP     #_Count_20_MSW          :Point to the memory location
            SACL    _Count_20_MSW           :Zero the 20us count registar:
```

```
                LDP     #_TachUpLimit           :Point to the memory location
                SACL    _TachUpLimit            :Zero the Tach Pulse check limit
                LDP     #_TachLowLimit          :Point to the memory location
                SACL    _TachLowLimit           :Zero the Tach Pulse check limit
                LDP     #_Stat_Buffer           :Point to the memory location
                SACL    _Stat_Buffer            :Zero the Status Buffer:
                LDP     #_Cmd_Bits              :Point to the memory location
                SACL    _Cmd_Bits               :Zero the Command Bits registar:
        :Reset should have cleared the Control Port. Now Clear the image.
                LDP     #_Ctrl_Image            :Point to the memory location
                SACL    _Ctrl_Image             :Zero Control Port memory image
        : Show that the Velocity Table has not been initialized
                LDP     #_Vel_Table             :Initialize Vel Table to -1
                SPLK    #-1._Vel_Table          :Say the table is not initialized
        : Show that the Inverse Time Table has not been initialized
                LDP     #_InverseTime           :Initialize Inverse Time to -1
                SPLK    #-1._InverseTime        :Say the table is not initialized
        : Initailize the Servo Loop Compensation Values
                LDP     #Focus_N1               :Point to the memory location
                SPLK    #Foc_N1.Focus_N1        :Store the initial value
                SPLK    #Foc_N2.Focus_N2        :Store the initial value
                SPLK    #Foc_N3.Focus_N3        :Store the initial value
                SPLK    #Foc_D2.Focus_D2        :Store the initial value
                SPLK    #Foc_D3.Focus_D3        :Store the initial value
                SPLK    #Foc_G.Focus_G          :Store the initial value
                SPLK    #Fin_N1.Fine_N1         :Store the initial value
                SPLK    #Fin_N2.Fine_N2         :Store the initial value
                SPLK    #Fin_N3.Fine_N3         :Store the initial value
                SPLK    #Fin_D2.Fine_D2         :Store the initial value
                SPLK    #Fin_D3.Fine_D3         :Store the initial value
                SPLK    #Fin_G.Fine_G           :Store the initial value
                SPLK    #Cr_N1.Crs_N1           :Store the initial value
                SPLK    #Cr_N2.Crs_N2           :Store the initial value
                SPLK    #Cr_N3.Crs_N3           :Store the initial value
                SPLK    #Cr_D2.Crs_D2           :Store the initial value
                SPLK    #Cr_D3.Crs_D3           :Store the initial value
                SPLK    #Cr_G.Crs_G             :Store the initial value
```

```
        SPLK    #Pn_G.Pin_G             :Store the initial value
        SPLK    #08000h.Sign_Bit        :Store the initial value
        SPLK    #0._FineDacZero         :Store the initial value
        SPLK    #0._CrsDacZero          :Store the initial value
        SPLK    #0._Foc_Err_Cnt         :Store the initial value
        SPLK    #07FFFH._Focus_Limit    :Store the initial value
        SPLK    #0._Fine_Err_Cnt        :Store the initial value
        SPLK    #07FFFH._Fine_Limit     :Store the initial value
        CLRC    INTM                    :Enable interupts
        RET                             :Return to the calling program
```